MASTER EN SCIENCES MATHÉMATIQUES

Les algorithmes génétiques: théorie et applications

HENDRICKX, Romain

Award date:
2011

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX
NAMUR

Faculté des Sciences

Les Algorithmes Génétiques : Théorie et applications

Mémoire présenté pour l'obtention

du grade académique de master en sciences mathématiques

Romain HENDRICKX

Juin 2011

Remerciements

Je tiens à remercier toutes les personnes ou organismes qui ont contribué à l'aboutissement de ce mémoire. Tout d'abord, je tiens à remercier mon promoteur de mémoire, Timoteo Carletti, pour la supervision de mon travail. Je tiens à le remercier autant pour ses qualités humaines que scientifiques. Merci à lui pour ses remarques pertinentes, le temps qu'il m'a consacré et pour sa grande disponibilité.

J'aimerais également remercier Anne Lemaître, Jean-Paul Leclercq et Bertrand Hespel qui constituent le jury de ce mémoire. J'espère qu'ils auront autant de plaisir à le lire que j'en ai eu en le réalisant.

Ensuite, j'aimerais remercier Suzanne Loret et Anne Lemaître d'avoir rendu possible mon stage à l'Université de Warwick en Angleterre, et qui a abouti à l'écriture du dernier chapitre de ce mémoire. Merci également à Thomas Nichols, mon maître de stage, de m'avoir accueilli pendant ces trois mois et de m'avoir consacré de son temps pour répondre à mes nombreuses questions. Merci également aux FUNDP pour le soutien financier qui a permis ce voyage.

J'aimerais également remercier l'ensemble des professeurs et assistants du département pour m'avoir donné le goût des mathématiques et pour l'influence positive qu'ils ont pu avoir sur moi.

Je tiens aussi à remercier mes parents pour le soutien permanent qu'ils m'ont procuré pendant l'ensemble de mes études. Merci pour leur tolérance et pour leur flexibilité pendant les périodes de "travail intense". Merci également à mes deux frères et ma soeur pour être ce qu'ils sont, et pour la relation privilégiée que j'entretiens avec eux.

Merci enfin à mes amis, rencontrés à l'université ou avant, pour leur sympathie, leur compréhension et pour tous les bons moments passés ensemble.

Résumé

Les algorithmes génétiques (GAs) sont des méthodes de résolution métaheuristiques pour les problèmes d'optimisation combinatoire, dont le fonctionnement est inspiré du processus d'évolution décrit par Darwin.

L'idée générale est de faire la comparaison entre les solutions admissibles du problème d'optimisation et un ensemble d'individus évoluant dans un monde abstrait, dans lequel leur adaptabilité est décrite par la fonction objectif, de telle manière à ce que plus la fonction objectif est grande et plus les individus sont "adaptés" à leur environnement. Dès lors, en partant d'une population initiale donnée et en simulant un processus d'évolution basé sur l'alternance d'opérateurs de variations, permettant d'explorer l'espace des solutions admissibles, et d'un opérateur de sélection, permettant de ne garder que les meilleurs individus (i.e. ayant la plus grande fitness), les algorithmes génétiques permettent d'obtenir au fil des générations un ensemble d'individus de plus en plus "adaptés à leur environnement", et donc par construction ayant une valeur objective de plus en plus proche de la solution optimale.

Le but de ce mémoire est de donner un aperçu du fonctionnement général des GAs. Une première partie du mémoire est consacrée à l'introduction de la structure générale des GAs, avec les spécifications les plus courantes pour les opérateurs de variation et de sélection (chapitre 1 et 2), ainsi qu'une justification théorique de leur utilisation (chapitre 3). La deuxième partie du mémoire est consacrée à l'application des GAs sur certains problèmes d'optimisation combinatoire. Nous étudions par exemple le problème de stratégie optimale d'un robot dont la tâche est de nettoyer une surface recouverte de canettes vides (connu sous le nom de "Robby, the Soda-Can-Collecting Robot", chapitre 4). Nous considérons également l'optimisation robuste des expériences par événements dans le cadre de l'imagerie par résonnance magnétique fonctionnelle (réalisé dans le cadre du stage de Master 2, chapitre 5).

Mots-clés : Algorithmes Génétiques, optimisation combinatoire, métaheuristique, théorème des schémas, Imagerie par Résonance Magnétique fonctionnelle (IRMf), expériences par événements, robustesse.

Abstract

Genetic Algorithms (GAs) are metaheuristics for discrete optimization problems inspired by Darwin's theory of evolution.

Roughly speaking, candidate solutions from the search space are seen as individuals from an abstract world, in which their adaptibility is described by some fitness function corresponding to the objective function, such that higher is the objective value, fitter is the individual. Then, considering some initial random-created individuals and simulating on this population an evolution process, based on both recombination operations, whose goal are to introduce perturbations in candidate solutions in order to explore the search space, and selection operation, whose goal is to determine, according to the individual's fitness, which of them are chosen to the next generation, GAs provide individuals with fitness growing up over generations, and thus candidate solutions closer to the optimal value.

The main goal of this master's thesis is to give an overview of how and why GAs works. The first part consists in introducing the basic structure of a genetic algorithm, and giving some well-known specifications for the recombination and selection operators (chapter 1 and 2), as well as a theoretical justification of their use (chapter 3). The second part is devoted to the illustration of the GAs on a few concrete and/or real problems. For example, we study the problem of designing an optimal strategy for a robot cleaning a given floor strewn with empty soda cans (known as "Robby, the Soda-Can-Collecting Robot", chapter 4). We also consider the robust optimization of event-related design in fMRI (achieved in the framework of the research stage of the Master 2, chapter 5).

Keywords : Genetic Algorithms, combinatorial optimization, metaheuristics, Schema theorem, functional Magnetic Resonance Imaging (fMRI), event-related designs, robustness.

Table des matières

Chapitre 1

Introduction

Ce chapitre a pour but de fixer le cadre d'application et de motiver l'utilisation des algorithmes génétiques. Dans la section 1.1, nous présenterons le domaine de l'optimisation combinatoire. Nous donnerons ensuite quelques exemples célèbres de problèmes combinatoires dans la section 1.2. Nous verrons dans la section 1.3 pourquoi la plupart de ces problèmes sont considérés comme "difficiles". Nous parcourrons enfin dans la section 1.4 les principales méthodes de résolution de ce type de problèmes, dont font notamment partie les algorithmes génétiques.

1.1 Optimisation combinatoire

De nombreux secteurs de l'industrie et de la recherche sont confrontés de nos jours à la difficulté de résolution de problèmes très complexes [1], tels le placement de composants électroniques, la gestion du trafic aérien ou automobile, la planification d'horaires, l'ajustement de paramètres d'un modèle numérique de processus, etc. Ces problèmes peuvent souvent s'exprimer sous forme d'un *problème d'optimisation*, qui consiste à définir une fonction, appelée *fonction objectif*, qui dépend d'un ensemble de variables propres au problèmes, et que l'on veut minimiser ou maximiser par rapport à cet ensemble de variables.

De manière tout à fait générale, un problème d'optimisation peut être décrit sous la forme suivante :

Problème 1.1 (Problème général d'optimisation)**:**
Étant donné une fonction $f : X \to \mathbb{R}$ d'un ensemble X, appelé *ensemble des solutions admissibles*, dans l'ensemble des réels, rechercher un élément $x \in X$ qui minimise ou qui maximise f sur X.

1. Nous donnerons une définition plus rigoureuse de la complexité des problèmes dans la section 1.3.

Nous noterons un tel problème respectivement

$$\begin{cases} \min f(x) \\ \text{s.c. } x \in X, \end{cases} \qquad \text{ou} \qquad \begin{cases} \max f(x) \\ \text{s.c. } x \in X. \end{cases} \tag{1.1}$$

Notons qu'un problème de maximisation peut se ramener à un problème de minimisation, et vice-versa, grâce à l'identité suivante :

$$\begin{cases} \max f(x) \\ \text{s.c. } x \in X \end{cases} \iff \begin{cases} -\min\left(-f(x)\right) \\ \text{s.c. } x \in X \end{cases}$$

On peut donc considérer l'une ou l'autre forme de manière équivalente. C'est pourquoi nous ferons le choix dans l'ensemble de ce mémoire, et ce sans perdre aucune généralité, de ne considérer que les problème de *maximisation*. Ce choix s'est imposé dès le début dans ce domaine pour respecter les fondements de la théorie de l'évolution sur laquelle est basé le fonctionnement général des algorithmes génétiques, selon lequel une population donnée évolue de manière à *maximiser* une certaine fonction de *fitness* de ses individus [2].

Dans l'ensemble des problèmes d'optimisation, on peut distinguer les problèmes à variables continues et les problèmes à variables discrètes. Dans le premier cas, les variables peuvent prendre des valeurs dans des ensembles non dénombrables, typiquement des valeurs réelles. Dans le second cas, les variables ne peuvent prendre qu'un nombre dénombrable de valeurs. On parle aussi d'optimisation *combinatoire*. Dans la plupart des problèmes combinatoires, cet ensemble possède un très grand nombre d'éléments mais est *fini*.

Même s'ils ont été adaptés par la suite à l'optimisation continue, les algorithmes génétiques ont été développés à la base dans le cadre de l'optimisation combinatoire. C'est pourquoi nous nous placerons exclusivement dans ce cadre d'étude dans la suite du travail. Nous ne considérerons donc que le cas où l'ensemble des solutions admissibles X est fini ou dénombrable.

Il est important de distinguer ces deux catégories de problèmes car ceux-ci diffèrent (notamment) par le type des méthodes [3] pouvant être appliquées pour les résoudre. Dans le cas des problèmes continus, il existe une panoplie de méthodes *classiques*, qui utilisent pour la plupart les dérivées de la fonction objectif, et dont la convergence globale n'est souvent assurée que sous certaines hypothèses sur la structure du problème, par exemple la convexité de la fonction ou des contraintes. Pour les problèmes combinatoires, la fonction objectif ne possède pas de dérivées, et les méthodes classiques ne peuvent

2. La définition complète et rigoureuse des algorithmes génétiques sera donnée dans le chapitre 2.

3. Nous nous plaçons ici dans le cadre de l'optimisation numérique, *i.e.* faisant appel à des méthodes de résolution numérique.

pas être appliquées. Il faut donc avoir recours à d'autres méthodes, que nous verrons dans la section 1.4. Notons que parfois certains problèmes continus sont également traités par des méthodes d'optimisation combinatoire, par exemple lorsque la dérivée est trop "chère" à calculer. Mais avant d'aborder les méthodes de résolution des problèmes d'optimisation combinatoire, donnons en quelques exemples.

1.2 Exemples

La meilleure manière de se rendre compte de ce que sont les problèmes combinatoires est sans doute d'en donner quelques exemples. Cette section présente deux exemples emblématiques de problèmes combinatoires : le problème du voyageur de commerce et le problème de coloration des graphes. Ceux-ci sont intéressants car ils sont à la fois simples à formuler, et très difficiles à résoudre quand la taille [4] du problème devient grande. C'est pour cette raison et parce qu'ils ont de multiples applications pratiques qu'ils représentent des problèmes "références", qui ont été et qui sont utilisés sans cesse lors du développement de nouvelles méthodes d'optimisation combinatoire, pour permettre une comparaison de l'efficacité et des performances [5] de celles-ci avec les méthodes existantes. Notons qu'il existe de nombreux autres problèmes paradigmatiques en optimisation combinatoire, notamment les problèmes du "couplage minimal de points", de l'"affectation quadratique", de la "satisfiabilité propositionnelle", etc.

1.2.1 Le problème du voyageur de commerce

Le problème du voyageur de commerce (en anglais : Travelling Salesman Problem, ou TSP) est celui rencontré par un représentant qui doit visiter un nombre donné de clients situés dans différentes villes en empruntant le plus court chemin possible, en respectant la contrainte de ne pas passer deux fois par la même ville et en terminant son tour dans la ville de départ. Une instance de ce problème ainsi que sa solution optimale sont représentées sur la figure 1.1 ; c'est le cas particulier où les villes à visiter sont les chefs-lieux des provinces de Belgique.

On peut modéliser le problème du voyageur de commerce de manière plus formelle et plus abstraite grâce à la théorie des graphes. Pour cela, rappelons la notion de cycle Hamiltonien d'un graphe. Attention que nous ne rappelons pas ici toutes les notions de la théorie des graphes, car nous supposons

4. La taille d'un problème n'est pas une notion univoque. Intuitivement, c'est le nombre de données nécessaires pour représenter le problème. Par exemple, dans le cas du voyageur de commerce, cela correspondra au nombre de villes, et dans le cas de la coloration d'un graphe, au nombre de sommets du graphe.

5. La notion de l'efficacité d'un algorithme sera développé dans la section 1.3.

FIGURE 1.1 – Représentation d'une instance du problème du voyageur de commerce, et de sa solution optimale (ligne rouge). Source http ://www.tsp.gatech.edu/maps/.

que le lecteur en a des connaissances basiques. Le lecteur intéressé pourra par exemple consulter la référence [Fournier, 2009], qui introduit les bases de la théorie des graphes ainsi que ses applications, et en particulier celles qui relèvent de l'optimisation combinatoire.

Soit $G := (V, E, w)$ un graphe pondéré et orienté, où V représente l'ensemble des sommets de G, $E \subseteq V \times V$ l'ensemble des arcs de G et $w : E \to \mathbb{R}$ une fonction de poids sur G, qui associe à chaque arc $e \in E$ un poids $w(e)$. Définissons d'abord les notions de *chaîne*, de *cycle* et de *cycle hamiltonien* dans le graphe G.

1. Une *chaîne* de G est une liste $(u_1, \ldots, u_k)$ de sommets de G telle que chaque couple $(u_i, u_{i+1}), i = 1, \ldots, k-1$, est un arc de G.
2. Un *cycle* est une chaîne dont le premier et le dernier sommet coïncident.
3. Un *cycle Hamiltonien* est un cycle qui visite chaque sommet de G exactement une fois, sauf le point de départ. Autrement dit,

$$c = (u_1, u_2, \ldots, u_n, u_1)$$

est un cycle Hamiltonien si $\{u_1, u_2, \ldots, u_n\} = V$, où $n = \sharp V$.

On peut alors définir la fonction de poids w sur l'ensemble des chaînes de G. Soit $c := (u_1, \ldots, u_k)$ une chaîne de G. Le poids de c est défini par la somme des poids de chacun des arcs qui la composent :

$$w(c) := \sum_{i=1}^{k-1} w((u_i, u_{i+1})).$$

Remarquons que dans les définitions précédentes, nous permettons aux arcs et aux chaînes d'avoir éventuellement un poids *négatif*. Certaines applications requièrent en effet une modélisation dans laquelle certains arcs possèdent des poids négatifs. C'est par exemple le cas de problèmes faisant intervenir des *coûts monétaires* ; dans ce cas les poids négatifs représentent des *gains*, alors que les poids positifs représentent des *dépenses*. Rechercher un cycle Hamiltonien de poids minimal consiste alors à rechercher un tour permettant de minimiser les dépenses ou, de manière équivalente, maximiser les gains. Signalons enfin que nous permettons la présence de cycles dont le poids total est négatif. Une telle définition serait problématique dans le cas où l'on considère un problème pour lequel il n'est pas proscrit de passer plusieurs fois par le même sommet, et donc en particulier plusieurs fois par un même cycle. En effet, dans ce cas le minimum vaut $-\infty$ et consiste à passer indéfiniment par un cycle de poids négatif. Cependant, pour le TSP, la contrainte de passer exactement une fois par chaque sommet empêche un tel cas de figure, et la présence de cycles de poids négatif ne pose pas de problème.

Le TSP peut maintenant être défini formellement de la manière suivante :

Problème 1.2 (TSP)**:**
Étant donné un graphe pondéré et orienté G, le problème du voyageur de commerce consiste à trouver un cycle Hamiltonien de poids minimal.

Notons enfin que le TSP intervient dans de nombreuses applications réelles, comme la création de puces électroniques, le séquençage de l'ADN, dans certains problèmes de planification, etc.

1.2.2 La coloration de graphes

La coloration d'un graphe est le fait d'attribuer une couleur à chacun de ses sommets, de telle manière à ce que les sommets adjacents n'aient pas la même couleur. Le problème de minimisation y étant associé est la recherche d'une coloration permettant de minimiser le nombre de couleurs différentes utilisées. Ce nombre minimal de couleurs est appelé le *nombre chromatique* du graphe considéré.

Avant de définir plus rigoureusement le problème de coloration d'un graphe, notons tout d'abord que ce dernier est impossible lorsque le graphe possède des sommets qui sont reliés à eux-mêmes (des boucles). En effet, deux sommets reliés par une arête doivent être marqués d'une couleur différente, ce qui est évidemment impossible lorsque ces sommets coïncident.

Ensuite, dans ce problème, il est surtout important de savoir quels sommets sont reliés entre eux, mais le nombre d'arêtes les reliant, ou le "sens de liaison" ne jouent aucun rôle. C'est pour cela que la coloration de graphes n'est définie que pour des graphes non-dirigés, sans boucle ni arête multiple.

Remarquons enfin qu'avec la formulation du problème que nous avons proposée, il n'existe jamais de solution unique au problème, sauf dans le cas d'un graphe à un sommet. En effet, considérons une solution optimale qui utilise n couleurs différentes. Dès lors, toute permutation de ces n couleurs fournit une solution optimale différente. Les couleurs ne sont donc intéressantes que dans le partitionnement des sommets du graphe qu'elles définissent. Une notion mathématique plus adaptée au problème de coloration des graphes est la notion de *stable* d'un graphe.

Pour cela rappelons d'abord la définition de *sous-graphe*. Soit $G := (V, E)$ un graphe non-dirigé. $G' := (V', E')$ est un *sous-graphe* de G si et seulement si G' est un graphe tel que $V' \subseteq V$ et $E' \subseteq E$. Un *stable* S de G peut alors être défini comme étant un sous-graphe de G dont les sommets sont deux-à-deux non-adjacents.

Dès lors, la coloration d'un graphe peut être définie de la manière suivante :

Problème 1.3 (Coloration de graphes)**:**
Soit $G := (V, E)$ un graphe simple.
Une coloration de G est une partition de ses sommets en stables.
Le problème d'optimisation associé est celui de trouver une coloration de nombre de stables minimum.

Un exemple de la coloration d'un graphe est représenté sur la figure 1.2. Il s'agit du graphe de Petersen, coloré à l'aide de trois couleurs différentes. Ce problème particulier est très simple à résoudre, mais certaines instances du problème sont quant à elles très difficiles à traiter.

Signalons finalement que, comme pour le TSP, de nombreux problèmes peuvent être formulés sous forme de coloration de graphes comme par exemple l'allocation de registres, la planification d'horaires, ou encore le célèbre jeu du *sudoku*.

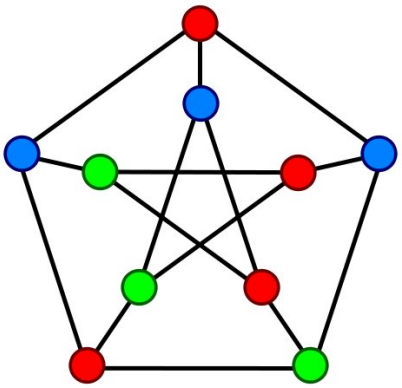


FIGURE 1.2 – Une coloration du graphe de Petersen avec trois couleurs. Source : http ://en.wikipedia.org/wiki/Graph_coloring.

1.3 Complexité des algorithmes et des problèmes

De nombreux problèmes combinatoires sont souvent qualifiés de *complexes*, et c'est cette complexité qui nécessite l'emploi de méthodes particulières comme les algorithmes génétiques. Cette section a pour but de préciser les notions de *complexité* pour les algorithmes d'une part et pour les problèmes d'autre part.

Pour motiver notre démarche, considérons la méthode la plus simple et la plus intuitive pour résoudre n'importe quel problème combinatoire : l'*énumération exhaustive*. Cette méthode consiste simplement à parcourir l'ensemble des solutions admissibles, évaluer la valeur de la fonction objectif en chacune d'entre elles, et retenir celle qui possède la valeur minimale.

Cependant le nombre de solutions admissibles croît généralement au moins de manière exponentielle avec la taille du problème. Par exemple, pour le TSP, on peut montrer facilement que le nombre de cycles Hamiltonien pour un graphe complet (*i.e.* un graphe dont tous les sommets sont adjacents) à n sommets est $\frac{(n-1)!}{2}$. Par conséquent, l'approche naïve de l'énumération exhaustive est simplement irréaliste en pratique. En effet, pour donner une idée du temps que prendrait cette méthode sur le TSP, supposons être en possession d'une machine qui peut réaliser 10^6 évaluations de la fonction objectif chaque seconde. Les temps approximatifs requis par une telle machine pour renvoyer une solution optimale pour différentes tailles du problème sont repris dans le tableau 1.3. Nous remarquons qu'à partir d'instances de taille $n = 18$, ce qui n'est pourtant pas très grand, cela devient tout simplement impossible, car il faudrait plusieurs années pour obtenir un résultat.

$n = 12$	$n = 15$	$n = 16$	$n = 17$	$n = 18$	$n = 20$	$n = 30$
19.9 s	12.1 h	7.56 j	121.08 j	5.63 ans	19287 ans	2.8 siècles

TABLE 1.1 – Temps pris par la méthode d'énumération exhaustive pour différentes tailles de problèmes, sur une machine capable de calculer 10^6 évaluations de la fonction objectif par seconde.

Donc, malgré la capacité théorique de l'algorithme à renvoyer une solution optimale, on ne peut pas s'en servir en pratique pour des instances de "trop grande taille". L'objectif de l'optimisation combinatoire est donc de développer des algorithmes permettant de résoudre les problèmes combinatoires *en un temps raisonnable*, c'est-à-dire des algorithmes qui soient *efficaces*. Une manière d'étudier l'efficacité d'un algorithme est de calculer sa *complexité algorithmique*. La section suivante présente les notions élémentaires de la théorie de la complexité algorithmique.

1.3.1 Complexité des algorithmes

Sur un ordinateur donné, l'exécution d'un algorithme demande certaines ressources en mémoire et en temps. En effet, lorsqu'un algorithme s'exécute sur (une instance d')un problème donné, chaque opération effectuée par celui-ci demande un certain *temps de calcul* au niveau du processeur. De même, de la mémoire doit lui être réservée dans l'ordinateur pour permettre la manipulation et le stockage des données et des résultats du problème considéré. L'analyse de la complexité d'un algorithme permet de quantifier cette nécessité. Nous ne considérerons cependant ici que la complexité en temps, car il constitue souvent le principal critère d'efficacité d'un algorithme. En effet, avec la mémoire dont disposent les ordinateurs actuels, les algorithmes posent plus rapidement problème par rapport à leur temps d'exécution que par rapport à la quantité de mémoire qu'ils requièrent. Nous désignerons donc simplement par *complexité* la complexité en temps de calcul.

Lors de l'exécution d'un programme, la lecture et l'exécution de chacune des instructions coûte un certain temps. Le calcul de la complexité d'un algorithme pourraît consister, comme nous l'avons fait pour analyser l'efficacité de la méthode de l'énumération exhaustive, à calculer le temps en secondes pris par l'algorithme pour certaines instances et sur une machine donnée. Cependant, même si une telle analyse peut être intéressante pour se "donner une idée" du temps pris par l'algorithme, elle n'apporte aucune information si on décide de changer de machine ou de données. C'est pour cette raison que la complexité algorithmique est définie de manière moins concrète, mais plus générale grâce à la notion d'*opération élémentaire*. Une opération élémentaire est une opération "simple", un "pas" de l'algorithme, comme par exemple une opération arithmétique, une affectation de valeur,

ou une comparaison. Une mesure de la complexité d'un algorithme peut alors être fournie par le calcul du nombre d'opérations élémentaires effectuées par celui-ci, en fonction de la taille des données.

La formalisation de cette idée peut être donnée de la manière suivante. Soit A un algorithme permettant de résoudre une classe de problèmes Π.

1. Le *coût de l'algorithme A pour l'instance* $\pi \in \Pi$, noté $\phi_A(\pi)$, est le nombre d'opérations élémentaires effectuées par A pour résoudre π.
2. La *complexité c_A de l'algorithme A* est la fonction définie sur l'ensemble des entiers naturels $\mathbb{N}$ par

$$c_A(n) := \max\{\phi_A(\pi) : \pi \in \Pi \text{ et } |\pi| = n\},$$

où $|\pi|$ désigne la taille de l'instance π.

Le maximum dans la définition de la complexité permet de déterminer le coût de l'algorithme A dans *le pire des cas*. On pourrait également étudier la complexité dans *le meilleur des cas* ou *en moyenne*. Cependant, la complexité dans le meilleur des cas n'apporte que peu d'informations quant à l'efficacité de l'algorithme, et la complexité en moyenne, bien qu'intéressante à ce point de vue, est généralement trop difficile à calculer.

Comme nous l'avons vu sur l'exemple de l'énumération exhaustive, le problème de connaître la complexité d'un algorithme se pose surtout lorsque la taille du problème devient grand. En effet, deux algorithmes ayant respectivement des complexités de n^2 et $2n^2$ ne diffèrent pas *significativement* lorsqu'on les compare à un algorithme de complexité 2^n, comme on peut le voir sur la figure 1.3.1.

Un outil qui permet de comparer les fonctions pour les grandes valeurs de leurs arguments est la notion d'*ordre de grandeur* d'une fonction. Soit $f : \mathbb{R} \longrightarrow \mathbb{R}$ une fonction à valeurs réelles. L'ensemble des fonctions du même ordre de grandeur que la fonction f, noté $\mathcal{O}(f)$, est défini par :

$$\mathcal{O}(f) := \{\, g : \mathbb{R} \longrightarrow \mathbb{R} \mid \exists k > 0 : \lim_{n \to +\infty} \frac{g(n)}{f(n)} \leq k\}.$$

Autrement dit, une fonction g est du même ordre que f si celle-ci peut être bornée supérieurement, pour n assez grand, par kf, avec $k > 0$. L'illustration de la notion d'ordre de grandeur d'une fonction est donnée sur la figure 1.4.

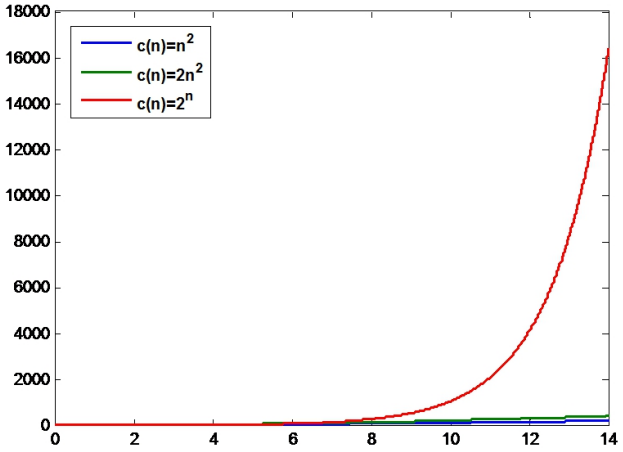


FIGURE 1.3 – Comparaison de la complexité d'algorithmes quadratiques et exponentiel.

L'ensemble des fonctions de même ordre que f peut donc être vu comme les fonctions qui "croissent moins vite" que f, à une constante près.

Donnons quelques exemples d'ordre de grandeur :

1. Les fonctions constantes sont $\mathcal{O}(1)$.
   En effet, soit $k \in \mathbb{R}$ une (fonction) constante.
   On a alors que $\lim_{n\to+\infty} \frac{k}{1} = k$, et donc $k \in \mathcal{O}(1)$.
2. Les polynômes de degré n sont $\mathcal{O}(x^n)$.
   En effet, soit $p(n) := a_n x^n + \cdots + a_1 x + a_0$ un polynôme de degré n.
   On a alors que $\lim_{n\to+\infty} \frac{p(n)}{x^n} = a_n$, et donc $p(n) \in \mathcal{O}(x^n)$.
3. $2n \in \mathcal{O}(n^2)$ car $\lim_{n\to+\infty} \frac{2n}{n^2} = 0$.
4. $2^n \in \mathcal{O}(2^{n+1})$ car $\lim_{n\to+\infty} \frac{2^n}{2^{n+1}} = \frac{1}{2}$.
5. $2^{n+1} \in \mathcal{O}(2^n)$ car $\lim_{n\to+\infty} \frac{2^{n+1}}{2^n} = 2$.
6. $n \in \mathcal{O}(n^2)$ car $\lim_{n\to+\infty} \frac{n}{n^2} = 0$.

La notion d'ordre de grandeur permet donc de grouper les algorithmes en *classes de complexité*. En effet, il suffit de regrouper dans une même classe les algorithmes ayant une complexité de même ordre. On parlera par exemple des algorithmes de complexité *linéaire* lorsque leur complexité $c(n) \in \mathcal{O}(n)$, ou de complexité *quadratique* lorsque $c(n) \in \mathcal{O}(n^2)$.

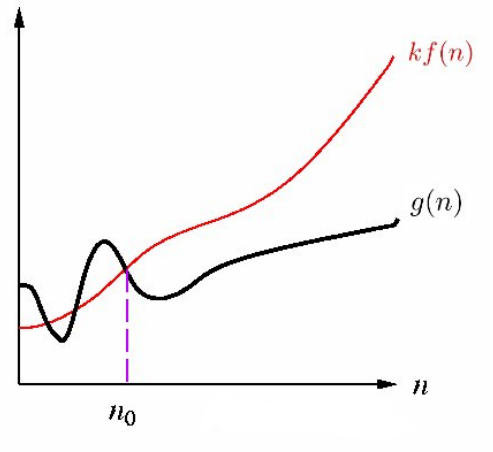


FIGURE 1.4 – Illustration de la notion d'ordre de grandeur d'une fonction. Source : http ://cc.ee.ntu.edu.tw/∼cchen/course/simulation/CAD/unit1A.pdf.

Voici les principaux ordres de grandeurs, et les relations qu'ils entretiennent ensemble :

$$\mathcal{O}(1) \subset \mathcal{O}(\log(n)) \subset \mathcal{O}(n) \subset \mathcal{O}(n\log(n)) \subset \mathcal{O}(n^k) \subset \mathcal{O}(l^n) \subset \mathcal{O}(n!),$$

où k et l désignent des constantes strictement plus grandes que 1.

Il est clair que plus un algorithme a une complexité proche de $\mathcal{O}(1)$ dans la chaîne d'implications précédentes, plus celui-ci sera considéré comme *efficace*. Mais la question est alors de se demander à partir de quelles classes de complexité les algorithmes peuvent être considérés comme *efficaces*. Pour tenter d'y répondre, nous avons réalisé une analyse analogue à celle effectuée pour la méthode de l'énumération exhaustive, c'est-à-dire nous avons calculé le temps d'exécution pour différentes classes de complexité et pour différentes tailles de problèmes, en supposant être ici en possession d'une machine capable d'effectuer 10^6 opérations fondamentales par seconde. Les résultats de ces calculs sont repris dans le tableau 1.2. Signalons que nous avons noté par le symbole "$+\infty$" les temps très grands (qui dépassent 1000 siècles).

Le tableau 1.2 permet de voir que les algorithmes dont la complexité est au moins exponentielle ne peuvent pas être appliqués en pratique sur des problèmes de grande taille. De plus, pour ces algorithmes, on peut remarquer que les temps calculés sont tellement grands à partir de certaines tailles de problèmes que même des améliorations considérables dans les performances des ordinateurs ne changeraient sans doute pas grand chose à l'affirmation précédente. C'est pour cette raison que la théorie de la complexité des algo-

	$n = 5$	$n = 10$	$n = 20$	$n = 50$	$n = 100$	$n = 1000$
1	1μs	1μs	1μs	1μs	1μs	1μs
$\log(n)$	1.6μs	2.3μs	2.99 μs	3.9μs	4.6 μs	6.9μs
n	5μs	10μs	20 μs	50 μs	100 μs	1 ms
n^2	25μs	100μs	400μs	2.5 ms	10 ms	1s
n^5	3.1 ms	0.1 s	3.2 s	5.2 m	2.77 h	36.7 ans
2^n	32 μs	1 ms	1.04 s	41.3 ans	$+\infty$	$+\infty$
10^n	0.1 s	2.77 h	$+\infty$	$+\infty$	$+\infty$	$+\infty$
$n!$	12 ms	3.62 s	881.8 siècles	$+\infty$	$+\infty$	$+\infty$

TABLE 1.2 – Temps pris par des algorithmes de différentes classes de complexité pour différentes tailles de données, sur une machine capable d'exécuter 10^6 opérations élémentaires par seconde. Les temps dépassant 1000 siècles sont notés "$+\infty$".

rithmes distingue particulièrement deux classes de complexité : la classe des algorithmes *polynomiaux*, et la classe des algorithmes *exponentiels*.

On définit les algorithmes *polynomiaux* comme les algorithmes dont la complexité en temps c_A est d'ordre de grandeur au plus polynomial, c'est-à-dire les algorithmes pour lesquels $\exists k \in \mathbb{N}$ tel que $c_A(n) \in \mathcal{O}(n^k)$. Les algorithmes ne vérifiant pas cette condition sont dits *exponentiels*.

Remarquons que cette définition est un peu abusive, car les algorithmes de complexité logarithmique sont considérés comme polynomiaux, alors que leur complexité n'est pas polynomiale. De même, un algorithme de complexité factorielle sera assimilé à un algorithme exponentiel, alors que sa complexité ne l'est pas non plus. Cette définition a surtout pour but de permettre la distinction des algorithmes qui sont *efficaces* de ceux qui ne le sont pas.

1.3.2 Complexité des problèmes

Maintenant que nous avons donné un critère d'efficacité pour les algorithmes, la question reste de savoir, pour un problème donné, s'il existe toujours un algorithme polynomial permettant de le résoudre. Lorsque ce n'est pas le cas, on peut concéder que la complexité doit être intrinsèque au *problème* considéré. Il semble donc intéressant de pouvoir porter l'étude de la complexité également sur les problèmes, et non plus seulement sur les algorithmes permettant de les résoudre. Une manière naturelle d'étudier la complexité d'un problème est de définir celle-ci comme la complexité de l'algorithme le plus efficace existant pour le résoudre. Dès lors, on peut généraliser la notion de classes de complexité aux problèmes ; un problème sera dans une classe de

complexité C s'il existe un algorithme permettant de résoudre ce problème qui appartienne à la classe C.

La notion de classes de complexité des problèmes permet donc de répartir ceux-ci selon leur niveau de "difficulté". De la même manière que l'on avait distingué les algorithmes polynomiaux et exponentiels pour définir l'efficacité de ceux-ci, on répartit les problèmes combinatoires en deux classes : la classe P et la classe NP[6].

La classe P est la classe des problèmes pouvant être résolus en un temps polynomial. Ceux-ci sont donc considérés comme l'ensemble des problèmes "faciles" à résoudre. Par contre, la classe NP est la classe des problèmes qui admettent un algorithme polynomial capable de tester si une solution donnée est *valide*. Autrement dit, la classe NP est la classe des problèmes pouvant être résolus en énumérant l'ensemble des solutions possibles (éventuellement en un temps exponentiel), et en testant chacune d'entre elle grâce à un algorithme polynomial. On a de manière évidente la relation suivante :

$$P \subseteq NP.$$

Cependant, l'inclusion dans l'autre sens $NP \subseteq P$ est moins évidente, et constitue un problème ouvert en informatique théorique, même si beaucoup s'accordent à croire que $P \neq NP$, et donc qu'il serait impossible de trouver des algorithmes polynomiaux permettant de résoudre les problèmes qui appartiennent à la classe NP.

Parmi la classe NP, on peut encore distinguer les problèmes les plus difficiles, qui seront alors appelés *NP-difficiles*, grâce à la notion de *réduction polynomiale*, que nous n'expliquerons pas en détail dans ce travail. Mais intuitivement, un problème est NP-difficile s'il est au moins aussi difficile que tous les autres problèmes dans NP. Par exemple, le TSP et la coloration de graphes sont des problèmes NP-difficiles.

Les problèmes difficiles sont donc les problèmes NP ou NP-difficiles, puisqu'aucun algorithme polynomial n'est connu pour les résoudre. Dès lors, les algorithmes existants permettant d'obtenir les solutions exactes des instances d'un problème difficile ne peuvent en pratique pas être appliqués pour résoudre les instances de grande taille, qui constituent cependant la plupart

6. En réalité, la théorie de la complexité des problèmes est basée sur l'étude de problèmes de *décision*, qui sont les problèmes qui posent une question admettant la réponse "oui" ou "non", et non les problèmes d'optimisation. Nous commettons donc certains abus dans cette présentation car nous ne voulons donner ici qu'une présentation simplifiée de la théorie de la complexité, plus pour donner une intuition de ce que sont les problèmes difficiles que pour vraiment entrer dans des détails théoriques.

des applications réelles. Il faut donc avoir recours à des méthodes particulières pour traiter ces problèmes. C'est l'objet de la section suivante.

Remarquons cependant que tous les problèmes d'optimisation combinatoire ne sont pas difficiles, et ce même pour certaines applications réelles. Par exemple, le *problème de plus court chemin* dans un graphe, qui consiste à rechercher la chaîne de poids minimal reliant deux sommets d'un graphe, et qui semble à première vue exiger une recherche dans un ensemble de solutions admissibles exponentiel avec la taille du problème, peut être résolu de manière efficace (en un temps quadratique) grâce à l'algorithme de Dijkstra ([Cormen et al., 2009], pp. 658 - 662).

1.4 Méthodes de résolution des problèmes difficiles

Comme, par définition, il n'existe pas d'algorithme efficace permettant de résoudre les problèmes difficiles, et qu'il serait d'ailleurs impossible d'en trouver dans le cas où $P \neq NP$, d'autres techniques de résolution doivent être employées pour traiter ces problèmes. Il est donc primordial de distinguer d'une part les méthodes *exactes*, qui garantissent l'obtention d'un minimum global en un temps exponentiel, et d'autre part les méthodes *inexactes*, qui permettent d'obtenir des résultats en des temps acceptables en se contentant de solutions *approchées*, c'est-à-dire de solutions qui ne sont pas forcément optimales, mais qui sont d'"assez bonne qualité". Nous verrons d'abord dans la section 1.4.1 que l'utilisation des méthodes exactes n'est cependant pas toujours proscrite. Nous présenterons ensuite dans la section 1.4.2 les principales méthodes de résolution inexacte, dont font notamment partie les algorithmes génétiques.

1.4.1 Méthodes exactes

Les techniques de résolution inexactes ont l'avantage d'être rapides, et sont donc indispensables pour résoudre un bon nombre de problèmes, mais possèdent l'inconvénient de ne pas toujours procurer une solution optimale. On préfère donc généralement utiliser une méthode exacte lorsque c'est possible, c'est-à-dire lorsque le temps pris par celle-ci est raisonnable. Nous allons considérer dans cette section des cas particulier où l'utilisation des méthodes exactes est possible.

Le cas le plus évident est lorsque la taille des instances du problème que l'on considère est petite, car alors les algorithmes exponentiels peuvent fournir des solutions en des temps raisonnables. Cependant, comme nous l'avons déjà mentionné auparavant, dans les applications réelles ce sont généralement les instances de grande taille qui sont considérées, et les méthodes exactes deviennent alors inapplicables.

Ensuite, en pratique il s'avère que lorsqu'on considère un problème Π, les applications réelles ne font parfois partie que d'une certaine sous-classe $\Pi' \subset \Pi$ qui, elle, possède un algorithme efficace permettant de résoudre les instances qu'elle contient. Dès lors, il est possible d'obtenir les solutions optimales des instances intéressantes du problème en des temps polynomiaux.

Un autre cas où l'on peut se permettre d'utiliser des algorithmes exacts provient du fait que la notion de difficulté des problèmes est définie à partir de la complexité algorithmique *dans le pire des cas*. En effet, certains algorithmes possèdent une complexité dans le pire des cas exponentielle, mais une complexité *en moyenne* polynomiale. On parle parfois dans ce cas d'algorithmes *pseudo-polynomiaux*. La définition dans le pire des cas ne reflète donc pas toujours de manière idéale le comportement de l'algorithme. Dans ce cas, les algorithmes concernés peuvent être appliqués en pratique sans aucun soucis, et ce même sur des problèmes de grande taille. C'est par exemple le cas de l'algorithme du simplexe pour la programmation linéaire dont la complexité théorique est exponentielle, mais qui s'est avéré avoir des temps d'exécution polynomiaux en la taille du problème pour la plupart des instances (voir par exemple [Cormen et al., 2009], pp. 843 - 897). Celui-ci est d'ailleurs utilisé en pratique dans de nombreux secteurs de l'industrie.

Enfin, signalons qu'il existe de nombreuses méthodes exactes plus ou moins sophistiquées qui sont plus performantes que les méthodes simples et intuitives permettant de résoudre un problème donné. De telles améliorations dans la performance des méthodes exactes permettent de pouvoir appliquer celles-ci à des instances de plus en plus grandes. Par exemple, des procédés de *séparation et évaluation* permettent d'augmenter la performance de la recherche exhaustive en prenant en compte les propriétés du problème pour éviter *a priori* l'énumération de mauvaises solutions. Pour le TSP, de telles méthodes permettent de résoudre des instances de plusieurs centaines, voire plusieurs milliers de sommets.

1.4.2 Méthodes inexactes

Lorsqu'on considère un problème combinatoire difficile qui ne peut pas être résolu grâce à un algorithme exact, une approche qui permet d'obtenir des résultats en des temps acceptables est de se contenter de solutions *approchées*, c'est-à-dire de solutions qui ne sont pas forcément optimales, mais qui sont d'"assez bonne qualité". Cependant, puisqu'on ne connaît a priori pas la valeur de la fonction objectif pour la solution optimale, il est impossible de quantifier la qualité relative de la solution. Pour connaître l'efficacité des méthodes d'approximation (et on ne parle plus ici seulement de l'efficacité par rapport au temps, mais aussi par rapport à la qualité de la solution), il faut

donc tester celles-ci sur des instances dont on connaît les solutions optimales, et comparer la qualité des solutions approchées aux solutions optimales. Une notion qui permet de quantifier la qualité d'une solution approchée, lorsque la solution optimale est connue, est le *degré de sous-optimalité*.

Soient Π un problème combinatoire et $\pi \in \Pi$ une instance de ce problème de solution obtimale $x^\star$. Soit également A un algorithme inexact pour le problème Π. Le degré de sous-optimalité de la solution $\tilde{x}$ retournée par l'algorithme A pour l'instance π est défini par

$$r(\tilde{x}) := \frac{f(\tilde{x})}{f(x^\star)} \text{ ou } r(\tilde{x}) := \frac{f(x^\star)}{f(\tilde{x})}$$

suivant que l'on considère un problème de minimisation ou de maximisation respectivement.

Dès lors, le but est de concevoir des algorithmes d'approximation permettant d'obtenir des solutions dont le degré de sous-optimalité est proche de 1.

Signalons que les algorithmes inexacts peuvent être en partie *stochastiques*, et donc ne pas renvoyer une même solution pour deux exécutions différentes. C'est pourquoi le degré de sous-optimalité est défini sur les *solutions* retournées par un algorithme donné, et non sur l'algorithme lui-même.

Dans certains cas, on peut garantir un certain niveau de qualité pour une classe particulière d'instances d'un problème. Par exemple, pour le TSP, les instances vérifiant l'*inégalité triangulaire*, c'est-à-dire la propriété :

$$\forall u, v, w \in E : w(u, w) \leq w(u, v) + w(v, w)\,,$$

peuvent être résolues par l'algorithme polynomial de Christofides avec un degré de sous-optimalité d'au plus $r = 1.5$ (voir par exemple [Fournier, 2009], pp. 226 - 230).

Cependant, dans beaucoup d'autres cas, on ne peut pas garantir mathématiquement un seuil maximal pour le degré de sous-optimalité, et c'est alors la confrontation empirique de la méthode sur les instances du problème considéré qui permet de se rendre compte de la qualité de la méthode. Dans ce cas, on parle de méthodes *heuristiques*.

Parmi les méthodes heuristiques, on distingue généralement deux classes : les heuristiques *spécialisées*, et les heuristiques *générales*, également appelées *métaheuristiques*.

Les heuristiques spécialisées sont des méthodes développées pour un problème *particulier* qui tiennent compte de la structure du problème et qui exploitent les propriétés particulières de celui-ci. Par exemple, une heuristique pour le TSP est l'*algorithme de plus proche voisin* (NNH, Nearest Neighbour Heuristic. Voir [Hoos and Stützle, 2005], pp. 368 - 369) qui consiste à obtenir une solution approchée en *construisant* un cycle Hamiltonien de la manière suivante : on choisit d'abord un sommet de départ de manière aléatoire, et ensuite, à chaque itération et jusqu'à l'obtention un tour complet, on rajoute un sommet au tour partiel en choisissant le sommet suivant comme celui dont l'arête qui le relie au dernier sommet rajouté est de poids minimal. Cette méthode très simple et très intuitive amène à des solutions d'assez mauvaise qualité, mais il existe un grand nombre d'heuristiques spécialisées qui permettent d'obtenir de très bonnes solutions.

Les métaheuristiques ne sont quant à elles pas conçues dans le but de résoudre un problème en particulier, mais sont plutôt des structures générales d'algorithmes, des stratégies de recherche, qui peuvent être adaptées et appliquées à une grande série de problèmes différents.

De manière générale, pour faire face à l'explosion combinatoire de l'ensemble des solutions admissibles, les métaheuristiques sont basées sur une exploration *partielle* de celui-ci, et comportent souvent des procédés stochastiques dans leur implémentation. On distingue principalement deux paradigmes d'explorations : d'une part les méthodes de *recherche locale*, qui, étant donné une configuration initiale dans l'ensemble des solutions admissibles, passe itérativement de candidat en candidat de manière à augmenter progressivement la valeur de la fonction objectif, dans le but de trouver au final une solution d'assez bonne qualité, et d'autre part les algorithmes de *recherche globale*, parfois appelées *méthodes à population*, qui manipulent un *ensemble de candidats* en parallèle, de manière à améliorer la qualité *globale* des candidats considérés au fur et à mesure des itérations. Précisons le fonctionnement général de chacun de ces paradigmes.

L'élément principal dans une recherche locale est la notion de *voisinage*, qui permet de structurer l'espace des solutions admissibles en déterminant quelles sont les configurations qui sont "accessibles" à partir de chacune des configurations de X. Formellement, un voisinage pour une instance π d'un problème combinatoire Π est une fonction V qui associe à chaque candidat x de l'ensemble des solutions admissibles X un ensemble de solutions voisines $V(x) \subset X \setminus \{x\}$. Remarquons qu'il faut imposer que les ensembles $V(x)$ soient significativement plus petits que X, car le but de la recherche locale est de pouvoir explorer de manière *efficace* chaque voisinage.

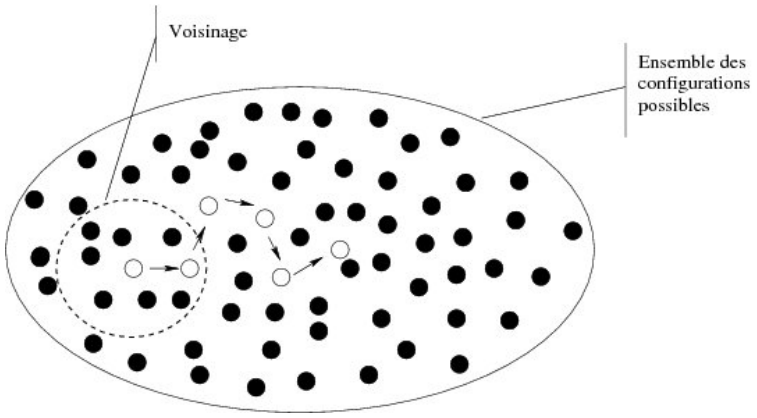


FIGURE 1.5 – Illustration du fonctionnement de la recherche locale. Source : [Lambert, 2006].

Une recherche locale consiste donc à partir d'une configuration initiale x_0 dans l'espace des solutions admissibles X, et à remplacer à chaque itération la configuration courante x par une configuration x' dans le voisinage $V(x)$, de telle manière à améliorer progressivement la qualité de la configuration courante, jusqu'au moment où l'on considère avoir obtenu une solution d'assez bonne qualité. La figure 1.5 illustre le fonctionnement d'une telle méthode de recherche dans l'espace des solutions admissibles.

Pour donner un exemple d'algorithme de recherche, présentons la *méthode de la plus forte montée*, qui est une généralisation au cas discret des méthodes de gradient pour les problèmes de maximisation continue.

Algorithme 1.1:
Choisir un élément initial $x \in X$ de manière aléatoire ;
Jusqu'à ce que $f(x) \geq f(x'),\ \forall x' \in V(x)$ **faire**
  Choisir $x := \text{argmax}\{f(s) : s \in V(x)\}$;
Fin
Retourner x.

L'algorithme de plus forte montée est donc l'algorithme qui choisit un candidat initial dans l'ensemble des solutions admissibles, et qui, à chaque étape de l'algorithme, choisit comme nouvel itéré le point qui maximise la fonction objectif f dans le voisinage du candidat courant, et ce jusqu'à ce qu'il ne soit plus possible d'améliorer la solution, c'est-à-dire jusqu'à l'obtention d'un maximum local.

Cependant, cette méthode ne conduit généralement pas au maximum *global* de la fonction objectif. Pour pouvoir s'"échapper" des maxima locaux, les méthodes de recherche locale doivent inclure une procédure de *diversification*, qui permette d'explorer une plus grande partie de l'espace des solutions admissibles. Dans le cas de la méthode de la plus forte pente, cela pourrait consister par exemple à "ré-initialiser" la méthode à partir d'un nouveau point aléatoire, pour obtenir un nouveau maximum local. Si la structure du problème particulier considéré ne possède qu'un faible nombre de maxima locaux, il suffirait de répéter ce processus un nombre assez grand de fois pour espérer obtenir le maximum global. Ceci dit, cette procédure augmente fortement le temps de calcul de la méthode, et il n'y a aucune garantie d'atteindre le maximum global grâce à ce procédé.

Pour s'extraire de manière plus efficace des maxima locaux, un processus de diversification qui est à la base des métaheuristiques de recherche locale les plus connues est la *détérioration temporaire* de la fonction objectif, qui permet de se déplacer plus largement dans l'espace des solutions admissibles. Une illustration de ce procédé est donnée sur la figure 1.6. Pour éviter la divergence de la méthode, chaque métaheuristique possède un mécanisme qui permet de contrôler ces détériorations. Un exemple de méthode de recherche locale est la méthode de *recherche tabou* (voir par exemple [Glover, 1989]), qui est inspirée du fonctionnement de la mémoire humaine à court terme. Un autre exemple est celui du *recuit simulé* (voir par exemple [Bertsimas and Tsitsiklis, 1993]), qui est inspiré quant à lui d'une analogie avec des phénomènes physiques en métallurgie.

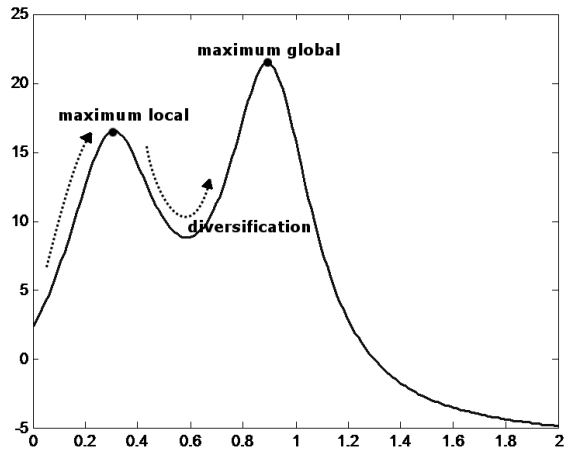


FIGURE 1.6 – Représentation du fonctionnement de la technique de diversification pour s'échapper d'un maximum local.

Considérons maintenant le cas des méthodes à population. Celles-ci sont donc des méthodes qui manipulent simultanément un ensemble de candidats, appelé généralement une *population* de candidats, et qui font interagir ceux-ci de telle manière à augmenter leur qualité au cours des itérations. Ce genre de méthode tire généralement son origine d'une comparaison avec des processus naturels. La manière dont les candidats "interagissent" entre eux pour améliorer la qualité globale est propre à chaque méthode. Par exemple, la méthode de *colonie de fourmis* (ACO, Ant Colony Optimization) est une méthode qui se base sur le comportement des fourmis dans une colonie, qui arrive à s'auto-organiser de manière complexe à partir de la collaboration d'un ensemble d'individus peu intelligents. Un autre exemple célèbre de méthode d'optimisation par population est l'*optimisation par essaims particuliers*, qui tire son origine de la comparaison avec le déplacement d'un groupe d'oiseaux, et qui se base également sur la collaboration entre individus pour arriver à un objectif commun. Enfin, le dernier exemple est celui qui nous intéressera dans la suite de ce mémoire, c'est celui des *algorithmes génétiques* (GAs, Genetic Algorithms), qui s'inspire de la théorie de l'évolution Darwinienne. Le chapitre suivant présente le fonctionnement général de ceux-ci.

Références pour le chapitre 1

Nous présentons ici les références principales utilisées pour l'élaboration de ce chapitre d'introduction. Tout d'abord, on retrouve l'influence constante de certains cours préalablement suivis, notamment les cours d'optimisation [Strodiot, 2009] et [Leclercq, 2010]. Ensuite, signalons que la présentation du TSP est tirée du livre [Hoos and Stützle, 2005], et la présentation du problème de coloration des graphes est inspirée de [Fournier, 2009]. En ce qui concerne la section sur la complexité, les références principales sont d'une part le cours d'algorithmique mathématique [Carletti, 2009] pour les notions et définitions, et le livre [Michel et al., 2002] pour l'introduction grâce au tableau 1.2. Enfin, la partie sur les méthodes de résolution des problèmes difficiles est un résumé de plusieurs sources : principalement [Leclercq, 2010], [Hoos and Stützle, 2005] et [Lambert, 2006].

Chapitre 2

Fonctionnement des algorithmes génétiques

2.1 Introduction

Souvent, les mathématiques sont utilisées comme un outil pour résoudre des problèmes réels. Typiquement, cela passe d'abord par une étape de modélisation du problème, où des hypothèses de simplification sont posées, ensuite par une étape d'analyse mathématique, où le problème dans sa forme mathématique idéalisée est résolu, et enfin par une étape finale qui consiste à dégager des conclusions sur le problème réel à partir de la solution mathématique trouvée.

Cependant, il est possible de faire l'inverse et de s'inspirer du fonctionnement de certains systèmes naturels comme base pour des outils de résolution de problèmes mathématiques. Nous pouvons citer les quelques exemples suivant : la méthode du *recuit simulé*, qui s'inspire d'un processus de fabrication d'alliages métalliques, les algorithmes de *colonie de fourmis*, qui s'inspirent du mode de communication des fourmis grâce aux phéromones, les *réseaux de neurones artificiels*, qui permettent de simuler l'apprentissage par le cerveau, etc. Les algorithmes génétiques font également partie de ces méthodes qui tirent leurs origines d'une telle comparaison avec un procédé naturel, et plus particulièrement de la comparaison avec le processus de l'évolution naturelle.

Le monde actuel est le fruit de plusieurs milliards d'années d'évolution, un processus basé sur la reproduction des espèces, sur la mutation et sur la sélection naturelle, permettant d'une part une diversification des espèces, et d'autre part la sélection des meilleurs individus, ou du moins les individus les mieux adaptés à leur environnement. Ce long processus a permis l'adaptation au cours du temps des espèce à leur milieu. Les algorithmes génétiques se basent sur le même principe : à partir d'un ensemble de candidats aléa-

toires, des opérateurs de reproduction et de sélection sont appliqués dans le but d'obtenir au cours des itérations des candidats de meilleure qualité.

Le premier à avoir eu une telle idée est J. D. Bagley dans sa thèse "The Behavior of Adaptive Systems Wich Employ Genetic and Correlative Algorithms" dans les années 50. Cependant, c'est J. H. Holland, un professeur de l'université de Michigan dans les années 60, qui est reconnu comme étant le fondateur des algorithmes génétiques. Il a été le premier à considérer un ensemble d'individus et à leur appliquer des opérateurs génétiques, mais toujours dans un cadre d'étude formel de populations. C'est l'un de ses élèves, D. Goldberg, qui a développé l'utilisation de tels procédés pour la résolution de problèmes mathématiques, et qui a donné naissance à ce qu'on appelle aujourd'hui communément les algorithmes génétiques.

2.2 Mécanisme général

2.2.1 Définitions et terminologie

Les algorithmes génétiques (GAs) s'inspirant du mécanisme de l'évolution, il n'est pas étonnant d'y retrouver des termes issus de la biologie. Il est donc opportun de donner préalablement un aperçu simplifié des éléments de biologie qui interviennent dans les algorithmes génétiques. Nous reprenons ci-dessous les principaux termes utilisés par les GAs, avec une explication (très) simplifiée de leur rôle biologique :

- Un *gène* est une séquence d'ADN. Il correspond au "support" d'une unité d'information génétique. Un exemple de gène est celui qui contient l'information génétique pour la couleur des yeux d'un individu [1].
- Un *allèle* est une version, une "valeur" possible pour un gène. Il correspond à l'information génétique contenue par un gène. Pour reprendre l'exemple du gène correspondant à la couleur des yeux, les allèles possibles pour ce gène sont *bleu*, *vert* et *brun*.
- Le *génotype* est l'ensemble de l'information génétique d'un individu. Il correspond à la composition des allèles de l'ensemble des gènes d'un individu.
- Le *phénotype* est l'ensemble des "caractères observables" (*i.e.* l'ensemble des caractéristiques anatomiques, morphologiques, moléculaires, physiologiques, etc.) d'un individu. Il est défini en opposition au génotype, qui n'est pas toujours suffisant pour déduire l'ensemble des caractéristiques d'un individu. En effet, celles-ci découlent de ses gènes, mais aussi d'une contribution de l'environnement extérieur à l'individu.

1. En réalité, la couleur des yeux est codée sur plusieurs gènes, mais nous avons choisi cette simplification pour faciliter la compréhension des différents termes.

- Un *chromosome* est le support de l'information génétique formé par un ensemble de gènes. Le génotype d'un individu consiste en un ensemble de taille fixée de chromosomes (par exemple, l'être humain possède un génotype de 46 chromosomes).
- La *fitness* est une mesure quantitative de la manière dont un individu d'un certain génotype est adapaté ou non au mileu dans lequel il évolue. C'est une mesure de la sélection naturelle.

Considérons maintenant le problème général d'optimisation 1.1. L'idée des algorithmes génétiques est d'assimiler l'espace des solutions admissibles X à un ensemble d'individus appartenant à un certain monde abstrait, et d'y appliquer un mécanisme d'évolution. De la même manière que, dans le monde réel, les processus évolutifs (par exemple la reproduction sexuée ou asexuée des individus, ou encore la sélection naturelle) ne se font pas directement sur les individus eux-mêmes mais sur l'information génétique qui les caractérise (leur génotype), les algorithmes génétiques manipulent une version codée des individus, typiquement des strings. C'est pour cette raison que nous définissons la notion de fonction de *codage* et de *décodage*.

Soit S un ensemble de strings de longueur n, et soit X l'ensemble des solutions admissibles d'un problème d'optimisation. Une fonction

$$\begin{array}{rccc} c: & X & \longrightarrow & S \\ & x & \longmapsto & c(x) \end{array} \tag{2.1}$$

et une fonction

$$\begin{array}{rccc} d: & S & \longrightarrow & X \\ & s & \longmapsto & d(s) \end{array} \tag{2.2}$$

sont respectivement appelées fonction de *codage* et fonction de *décodage* si :

1. la fonction d est injective ;
2. les fonctions c et d vérifient l'égalité $c \circ d = id_S$.

Les algorithmes génétiques ne manipuleront pas directement les candidats $x \in X$ mais leur représentation codée dans S. Les algorithmes génétiques ne traitent donc pas exactement le problème 1.1, mais une version codée de celui-ci :

Problème 2.1 (Problème d'optimisation encodé)**:**
Étant donné une fonction $f : X \to \mathbb{R}$ d'un ensemble X dans l'ensemble des réels, ainsi que des fonctions de codage et de décodage c et d sur un ensemble de strings S, rechercher un élément $s \in S$ qui maximise $f_S := f \circ d$ sur S.

Terme biologique	Algorithmes génétiques
Le génotype d'un individu $x \in X$	Représentation codée $c(x)$ de x dans S
Le phénotype d'un individu $x \in X$	Version non codée de x (dans X)
Un chromosome	N'importe quel string $s \in S$
Un gène dans un chromosome donné	Une certaine case i du string s correspondant
L'allèle d'un gène donné	La valeur $s(i)$ du gène considéré
La fitness d'un individu $x \in X$	La valeur $f(x)$ de la fonction objectif au point x

TABLE 2.1 – Tableau des correspondances entre les termes biologiques et les objets manipulés par les GAs.

Les fonctions de codage et de décodage doivent être spécifiées suivant les demandes du problème considéré. Plusieurs codages sont possibles pour un même problème, mais amènent souvent à des performances différentes. Par exemple, pour certains problèmes, le codage de Grey, qui est une variante du codage binaire, est plus adapté que ce dernier ([Caruana and Schaffer, 1988]). Il est donc important de définir correctement les fonctions de codage et de décodage suivant le type de problème donné.

Remarquons également que lorsque $d(S) \subset X$, où l'inclusion $\subset$ est prise stricte, la recherche se fait dans une sous-partie *propre* de X. Lorsque la solution optimale se trouve dans $X \backslash d(S)$, l'algorithme ne sait donc atteindre qu'une solution approchée de la solution optimale, et ce, même si l'algorithme est très performant. Ceci souligne de nouveau l'importance de bien définir les fonctions de codage.

Maintenant ces définitions posées, nous pouvons donner la correspondance entre les termes biologiques utilisés et les objets mathématiques manipulés par les algorithmes génétiques dans le tableau 2.1.

2.2.2 Schéma algorithmique général

Les algorithmes génétiques identifient donc l'ensemble des solutions admissibles X à un ensemble d'individus appartenant à un certain monde dans lequel leur adaptabilité est décrite par une fonction de fitness correspondant à la fonction objectif f. En partant d'une sous-population initiale d'individus, les algorithmes génétiques vont simuler un processus d'évolution par l'alternance de quatre opérations :

1. La *sélection*, qui permet de retenir un certain nombre d'individus pour la reproduction en fonction de leur fitness.
2. La *reproduction*, qui permet d'obtenir de nouveaux individus (les enfants) à partir des individus pré-sélectionnés (les parents) en mélangeant d'une certaine manière leur code génétique. Le but est d'obtenir

des enfants au moins aussi bons, et donc potentiellement *meilleurs*, que leurs parents. À noter que pour cela, il est nécessaire que le codage des individus soit bien choisi.

3. La *mutation*, qui permet de déformer légèrement le code génétique de certains individus. Cette opération a pour but d'amener des informations génétiques qui ne sont pas forcément présentes dans la population courante, mais qui pourraient amener à une amélioration potentielle, éventuellement à long terme. Du point de vue du problème d'optimisation, cela correspond à tenter d'éviter la stagnation dans un maximum local.
4. Le *passage à la génération suivante*, qui permet de calculer la nouvelle génération à partir de la génération précédente et des individus issus de la reproduction et de la mutation.

Ces opérations sont répétés jusqu'au moment où une certaine condition d'arrêt est satisfaite. L'espoir est d'obtenir par ce processus des individus de mieux en mieux adaptés à leur environnement au fil des générations et donc, par construction, un ensemble de solutions admissibles qui prennent des valeurs de f de plus en plus proches de la valeur objective optimale. Une illustration du mécanisme des algorithmes génétiques est donnée sur la figure 2.1, et la structure algorithmique générale est donnée par l'algorithme 2.1.

Algorithme 2.1:
$t := 0$;
Calculer une population initiale de génotypes B_0 ;
Tant que une condition d'arrêt n'est pas satisfaite **faire**

1. Sélectionner un ensemble de génotypes de B_t pour obtenir B_S ;
2. Reproduire les génotypes de B_S pour obtenir B_R ;
3. Muter certains génotypes de B_R pour obtenir B_M ;
4. Calculer la nouvelle génération B_{t+1} à partir de B_t, B_R et B_M.

Fin

Évidemment, il faut encore spécifier chacune des étapes de l'algorithme pour que celui-ci soit effectif. Un algorithme génétique est n'importe quel algorithme qui spécifie cette structure. Les sections suivantes décrivent les spécifications les plus connues pour chacune des étapes.

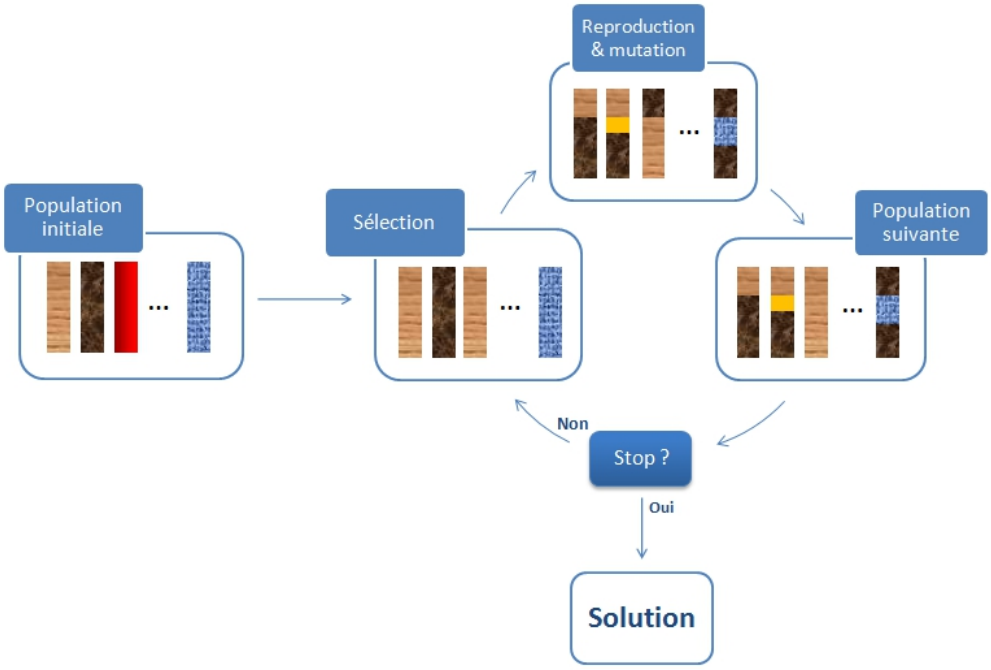


FIGURE 2.1 – Représentation du fonctionnement général d'un algorithme génétique.

2.3 Création d'une population initiale

La population initiale est généralement choisie de manière aléatoire dans S. Cela permet d'obtenir une bonne dispersion initiale, et d'ainsi éviter une convergence prématurée de l'algorithme.

Cependant, dans certains cas, il est préférable de choisir la population initiale de manière plus subtile pour augmenter les performances de l'algorithmes, que ce soit au niveau du temps de convergence ou de la qualité des solutions trouvées. Par exemple, certains choisissent parfois d'inclure dans la population initiale des candidats qui sont maxima locaux trouvés par des heuristiques spécialisées pour le problème donné. Ceci dit, lorsque c'est le cas, on préfère généralement choisir alors des heuristiques rapides, même si les solutions trouvées ne sont pas de très bonne qualité, car elles ne constitueront que le point de départ de l'algorithme génétique. Par exemple, pour le TSP, on pourrait considérer d'inclure dans la population initiale plusieurs solutions du NNH (cf. section 1.2.1).

2.4 Sélection

La sélection est le mécanisme dans le processus d'évolution qui permet de préférer les individus plus forts pour la reproduction, et qui mène ainsi au fil des générations à une meilleure adaptabilité des individus de la population. Il existe plusieurs spécifications de ce mécanisme, dont les plus connus sont la roulette (et ses variantes), le tournoi et l'élitisme. Développons chacune de ces stratégies de sélection.

2.4.1 Roulette

La sélection par roulette (*roulette wheel* en anglais) est la stratégie de sélection la plus classique. Elle consiste à sélectionner les individus de manière directement proportionnelle à leur fitness. Cela peut être accompli en réalisant un tirage au sort dans lequel chaque individu $b_{i,t}$ de la population[2] $B_t = \{b_{1,t}, \ldots, b_{m,t}\}$ à un certain temps t a une probabilité d'être choisi définie de la manière suivante :

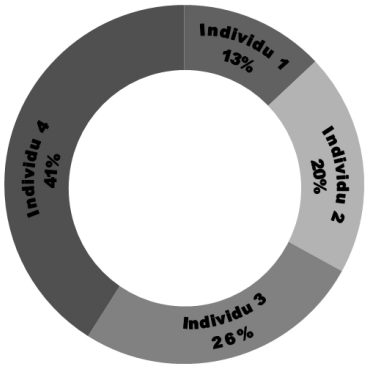
$$P(b_{i,t} \text{ est sélectionné}) = \frac{f(b_{i,t})}{\sum_{j=1}^{m} f(b_{j,t})}. \tag{2.3}$$

Remarquons que cette formule n'est valable que si la fitness est positive en chacun des individus $b \in B$. Si ce n'est pas le cas, il suffit d'appliquer une transformation non-décroissante $\phi : \mathbb{R} \to \mathbb{R}$ telle que $\phi(f(S)) \subset \mathbb{R}^+$, et de baser la sélection sur la fonction $\phi \circ f : X \to \mathbb{R}^+$.

Pour simuler de manière algorithmique un tel tirage au sort, on associe à chaque individu un sous-intervalle de $[0, 1]$ de longueur égale à sa probabilité. La sélection des individus peut se faire en tirant un nombre aléatoire dans $[0, 1]$ et en choisissant l'individu dont l'intervalle contient le nombre tiré. Ce procédé peut être comparé à une roulette où chacune des cases représente un individu de la population B et possède une largeur proportionnelle à la fitness de celui-ci. Une illustration d'une telle roulette est faite à la figure 2.2. Cette sélection est répétée autant de fois que le nombre d'individus que l'on veut sélectionner.

Cependant, ce type de sélection comporte l'inconvénient d'être fort sensible au choix de la fonction de fitness. En effet, si pour le problème de maximisation le fait de considérer la composée de la fonction objectif avec une fonction non-décroissante ne change pas les solutions optimales, cela a un impact direct sur les probabilités des individus. Par exemple, deux individus ayant respectivement une fitness de 100 et de 101 ont presque des

2. En toute rigueur, il faudrait faire la distinction entre les génotypes que manipulent les algorithmes génétiques et les individus auxquels ils correspondent. Cependant, comme la relation est directe grâce à la fonction de décodage, on se permet parfois de confondre les deux concepts.



Individu	Fitness	$\frac{f(b_i)}{\sum_j f(b_j)}$
1	33.54	0.13
2	51.60	0.20
3	67.08	0.26
4	105.78	0.41

FIGURE 2.2 – Représentation graphique de la sélection par roulette sur un exemple fictif d'une population de 4 individus dont les fitness respectives sont données dans le tableau de droite.

chances égales d'être tirés, mais si l'on considère la composée de la fitness avec la fonction $\phi(x) = x - 99$, *i.e.* la soustraction de la fitness de 99 unités, les chances deviennent respectivement $\frac{1}{3}$ et $\frac{2}{3}$, et donc la sélection avec l'une ou l'autre version de la fitness aura des effets totalement différents.

De manière générale, l'efficacité de la sélection dépend fortement de la variance de la fitness. En effet, lorsque la fitness a une variance trop petite, la sélection par roulette revient presque à un tirage au sort, alors qu'une variance trop grande entraîne une sélection trop "radicale" des meilleurs individus, et à une disparition trop rapide de la diversité des individus dans la population.

Pour pallier à cette sensibilité, une première approche est de considérer une fitness "standardisée", en retirant de la fitness sa moyenne sur l'ensemble des individus de la population, et en divisant le résultat obtenu par l'écart-type. Les fitness obtenues étant centrées autour de 0, on rajoute un terme constant assez grand (typiquement 3 fois la valeur de l'écart-type) que pour obtenir une fitness positive. Cette stratégie de sélection est appelée le *sigma-scaling*. Cependant, une telle définition de la fitness a pour inconvénient de dépendre de la population courante B_t. Une manière d'éviter cet inconvénient est de définir la nouvelle fitness comme

$$\tilde{f}(x) = \frac{f(x) - f_{\min}}{f_{\max} - f_{\min}}, \tag{2.4}$$

où $f_{\max}$ et $f_{\min}$ sont respectivement les valeurs maximale et minimale de la fonction objectif. Évidemment, ces valeurs ne sont pas toujours connues a priori. Il est donc souvent préférable de choisir une autre stratégie de sélection.

Une autre possibilité est de classer les individus par valeur de fitness croissante et de considérer comme nouvelle fitness le rang des individus dans ce classement. Sans surprise, cette stratégie est appelée la *stratégie par classement*.

2.4.2 Tournoi

Les deux solutions qui ont été proposées dans la section précédente consistent toutes deux à considérer une transformation de la fitness et à y appliquer ensuite la stratégie de sélection par roulette. Une stratégie qui n'applique pas un tel schéma de sélection proportionnelle et qui permet une non-dépendance de celle-ci avec les valeurs de la fitness est la sélection par tournoi.

Le tournoi est une stratégie qui consiste à tirer au hasard deux membres de la population et d'attribuer une probabilité p_T d'être choisi à celui qui a la plus grande fitness et une probabilité $1 - p_T$ à son adversaire, avec $p_T \in (\frac{1}{2}, 1)$ fixé. Ce processus est répété jusqu'à l'obtention de la population d'individus B_S.

2.4.3 Élitisme

L'élitisme est une stratégie de sélection optionnelle, qui est souvent couplée à l'une des méthodes vue précédemment. Cette stratégie consiste à réserver une partie de la population sélectionnée aux meilleurs individus de la population courante, empêchant que ceux-ci soient "perdus" à cause des procédés aléatoires présents dans les stratégies de sélection.

Il existe plusieurs manières de "garder les meilleurs individus". Si k places sont réservées pour les meilleurs individus, ceux-ci peuvent être choisis simplement comme les k meilleurs dans la population courante, ou bien être choisis par une stratégie de sélection parmi les $r \neq k$ meilleurs individus, etc. Il reste aussi à choisir si la sélection pour le reste des individus se fait sur l'entièreté de la population courante, ou sur la population sans les individus élites, etc. Beaucoup de variantes sont possibles, mais traduisent toutes l'objectif de préserver les meilleurs individus.

2.5 Reproduction

Considérons maintenant la reproduction, qui est l'opération qui permet le mélange de l'information génétique entre les individus sélectionnés. Comme pour la sélection, il existe un grand nombre de variantes, plus ou moins efficaces selon les cas.

L'opérateur le plus utilisé est le *crossover*, qui consiste à générer deux individus enfants en croisant les chromosomes de deux individus parents de telle manière à ce que les enfants possèdent aussi bien des gènes du premier parent que du deuxième.

Dans sa version la plus simple, le crossover consiste à couper les chromosomes parents à une position aléatoire, et d'ensuite inverser les queues des chromosomes pour obtenir les deux enfants. Ce type de crossover est appelé le crossover à un point. Il est souvent utilisé, mais il existe beaucoup d'autres versions (pour plus de détails, voir [Goldberg, 1989] ou [Geyer-Schulz, 1995]). Citons ici par exemple :

- *Le crossover à N_c points :* N_c points de coupure sont choisis aléatoirement, et une partie sur deux est inversée chez les deux parents.
- *Le crossover à nombre de points variable :* C'est un crossover à N_c points, où le nombre N_c de points de coupure est choisi aléatoirement.
- *Le crossover uniforme :* Chaque gène est permuté entre les deux parents avec une probabilité $\frac{1}{2}$.
- *Le shuffle crossover :* Le crossover à N_c points est appliqué sur une permutation aléatoire des parents, donnant des enfants sur lesquels est ensuite appliquée la permutation inverse.

Une illustration des opérateurs de reproduction est donnée à la figure 2.3.

Notons que les opérateurs de reproduction que nous avons présentés précédemment ne peuvent pas être appliqués à tous les problèmes. En effet, les crossovers classiques ne peuvent par exemple pas être appliqués au TSP, puisque ceux-ci produisent des enfants qui ne sont plus des cycles Hamiltoniens. D'autres opérateurs de reproductions ont donc été développés dans le but de mieux s'adapter à certaines classes de problèmes données. A titre d'illustration, citons par exemple le *partially matched crossover* (PMX, [Goldberg and Lingle, 1985]), l'*order crossover* (OX, [Davis, 1985]) et le *cycle crossover* (CX, [Oliver et al., 1987]) qui ont été créés particulièrement pour les problèmes où les solutions admissibles sont des cycles, comme le TSP.

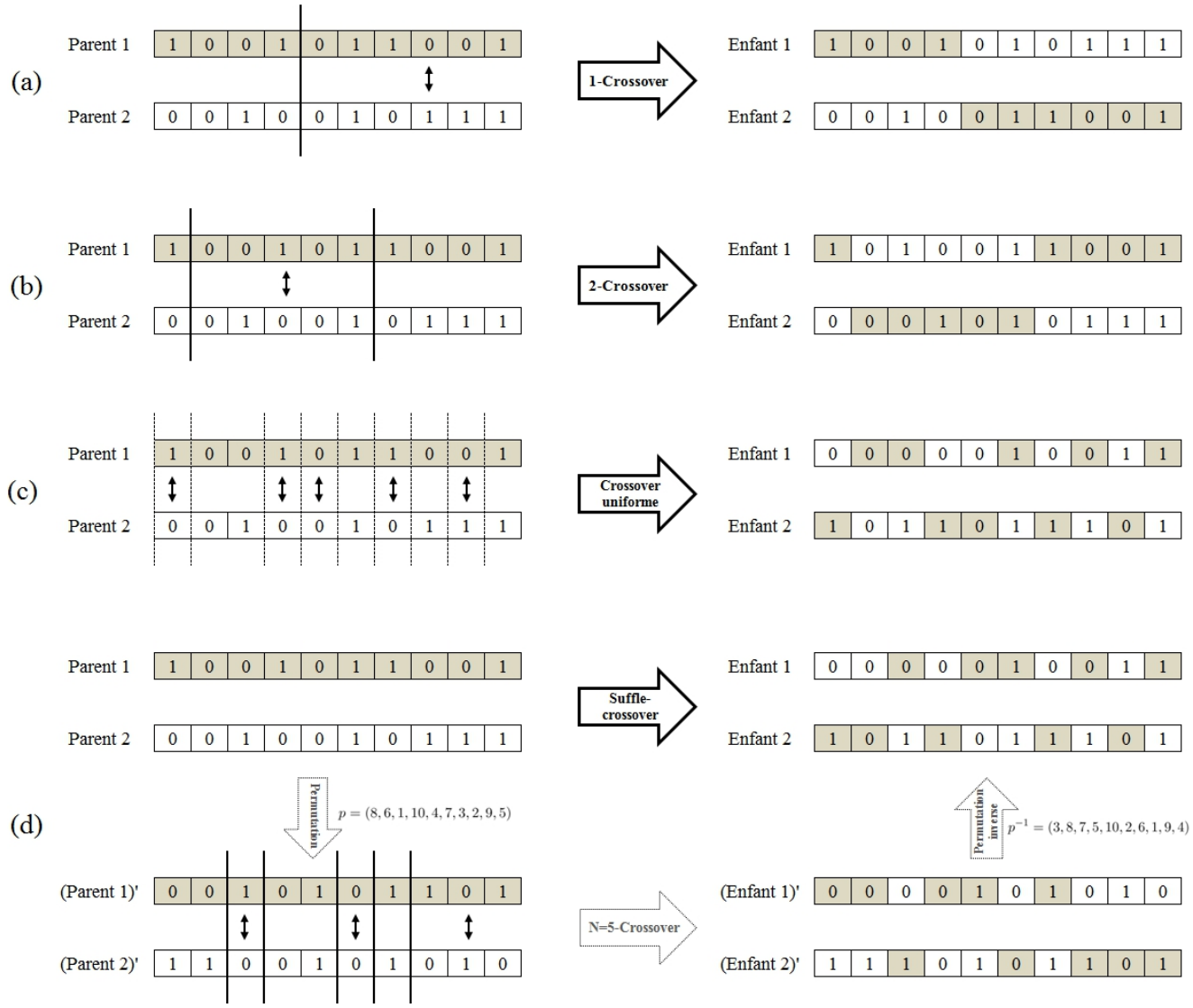


FIGURE 2.3 – Illustration des principaux opérateurs de crossover sur deux individus parents donnés de $S = \{0,1\}^{10}$. (a) : Crossover à 1 point de coupure ; (b) : Crossover à 2 points de coupures ; (c) : Crossover uniforme ; (d) Suffle crossover.

2.6 Mutation

Dans la nature, il arrive que le code génétique de certains individus soit légèrement modifié suite à un agent extérieur, comme par exemple l'exposition prolongée à certaines sources de radiation, ou suite à une petite erreur lors de la reproduction. Ces petites modifications sont rarement bénéfiques pour les individus (*i.e.* elles diminuent leur fitness), mais peuvent cependant amener une information génétique qui n'est présente dans aucun individu de la population et qui pourrait être avantageuse à long terme. Ce sont les mutations qui sont à l'origine des modifications évolutives. Elles correspondent, du point de vue des GAs, au mécanisme qui permet de s'échapper des maxima locaux.

Pour un individu donné, la mutation la plus simple est la $1-$*inversion*, qui consiste à choisir un gène au hasard et de changer son allèle par une valeur choisie aléatoirement dans l'ensemble des valeurs possibles pour le gène considéré.

À nouveau, il existe une kyrielle d'opérateurs qui permettent de "modifier légèrement" le code génétique des individus. Donnons les quelques exemples suivants de mutations possibles pour un individu donné (d'autres opérateurs de mutation peuvent être trouvés de nouveau dans [Goldberg, 1989] ou [Geyer-Schulz, 1995]).

- *La N_M-inversion :* Les allèles de N_M gènes choisis au hasard sont changés par des valeurs aléatoires.
- *L'inversion totale :* L'allèle de chacun des gènes de l'individu est changé par une valeur au hasard. Cet opérateur correspond à une N_M-inversion où $N_M = n$.
- *La sélection aléatoire :* Le génotype de l'individu est remplacé entièrement par un chromosome aléatoire.
- *L'inversion aléatoire :* L'allèle de chaque gène est remplacé avec une probabilité donnée.

Une illustration de ces opérateurs de mutation sont repris sur la figure 2.4.

Cependant, comme indiqué dans l'algorithme 2.1, la mutation n'est pas effectuée sur chacun des individus mais sur une partie d'entre eux seulement. Pour savoir quels individus muter, un paramètre p_M est fixé et définit la probabilité avec laquelle chacun des individus de B_R est muté.

Soulignons aussi que la mutation est un opérateur très important pour une bonne performance de l'algorithme. En effet, les opérateurs de sélection et de reproduction permettent de générer des individus dont les gènes permettent une meilleure adaptivité, mais cela engendre souvent une homogénéité croissante de la population, et donc la restriction de la recherche à

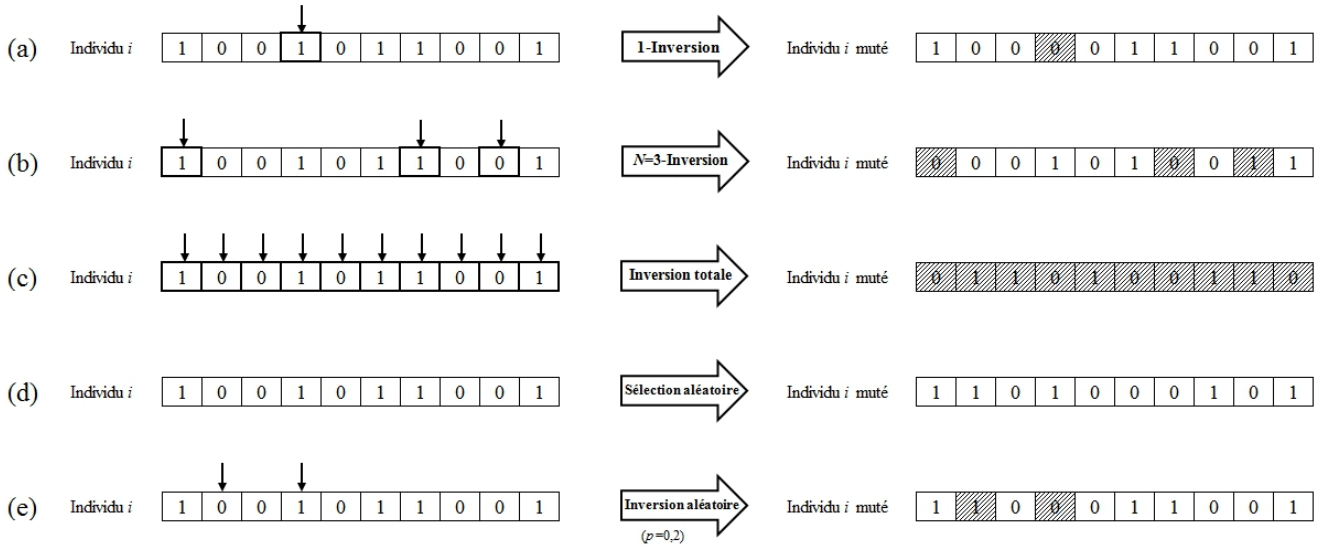


FIGURE 2.4 – Illustration des principaux opérateurs de mutation sur un individu donné de $S = \{0,1\}^{10}$. (a) : 1-Inversion ; (b) : N_M-Inversion, avec $N_M = 3$; (c) : Inversion totale ; (d) Sélection aléatoire ; (e) Inversion aléatoire.

un sous-espace de l'ensemble des solutions admissibles de plus en plus petit. Il permet donc, de temps en temps, d'incorporer des allèles qui n'étaient éventuellement plus représentés dans la population courante, et compense donc la perte d'information génétique due aux opérateurs de sélection et de reproduction.

Notons enfin que, comme pour le crossover, l'opérateur de mutation doit être adapté au type de problème considéré. Par exemple, certains opérateurs de mutation ont été définis de manière à respecter la structure de *cycle*, par exemple le *displacement mutation operator* (DM, [Michalewicz, 1992]), l'*exchange mutation operator* (EM, [Banzhaf, 1990]) ou encore le *insertion mutation operator* (ISM, [Michalewicz, 1992]).

2.7 Passage à la génération suivante

Considérons maintenant le pas 4 de l'algorithme 2.1, c'est-à-dire le passage à la génération suivante. Dans les trois pas précédents, le nombre d'individus produits par chaque opérateur n'est pas défini, et est fixé comme paramètre du programme. Mais le plus souvent, même si les ensembles issus de la sélection, de la reproduction et de la mutation n'ont pas le même nombre d'individus que la population courante B_t, le nombre d'individus par population, lui, reste fixe au cours des générations. Il faut donc déterminer une manière de sélectionner $\sharp B_t =: m$ individus parmi $\sharp B_S + \sharp B_R + \sharp B_M$.

La manière la plus simple est de fixer le nombre d'individus produits par chaque opérateur constant au nombre d'individus de la population. De cette manière, il suffit de choisir les individus B_M pour la génération suivante, après que ceux-ci aient été sélectionnés (pas 1), reproduits (pas 2) et mutés (pas 3).

Cependant, il existe de nombreuses autres choix de structure, suivant le nombre d'individus produits par chaque opérateur. On peut même réappliquer un opérateur de sélection (de type roulette ou tournoi) sur $B_t \cup B_S \cup B_R \cup B_M$. Le choix de la structure particulière est propre au programmeur, et il existe une infinité d'agencements et de paramètres différents pour chaque opérateur.

Notons que dans la littérature, le pas 4 n'est pas explicité clairement comme il en a été fait le choix ici, et souvent un choix arbitraire est fait a priori. De ce fait, même si un grand nombre d'articles abordent des comparaisons entre opérateurs de reproduction, de sélection ou de mutation, une comparaison sur la structure même de l'algorithme est rarement abordée. Nous avons choisi cette présentation pour rendre l'algorithme 2.1 le plus général possible.

2.8 Exemple de fonctionnement

Dans cette section on considère un problème simple et de petite taille pour illustrer le fonctionnement des algorithmes génétiques. Pour cet exemple, on utilise une spécification basique du schéma 2.1 sur l'espace des strings $S = \{0,1\}^n$ et à taille de population constante m :

Algorithme 2.2:
Soient donnés $m, n, t_{\max} \in \mathbb{N}_0$, avec m pair, ainsi que p_C et $p_M \in (0,1)$.

1. **Initialisation :** Créer une population initiale $B_0 = (b_{1,0}, \ldots, b_{m,0})$ de taille m, avec des individus choisis aléatoirement dans $\{0,1\}^n$;
2. **Corps de l'algorithme :** Pour $t = 0$ jusque $t_{\max}$:
   - 2.1. **Sélection :** sélectionner m individus $(b_{1,t+1}, \ldots, b_{m,t+1})$ dans B_t par la stratégie de la roulette ;
   - 2.2. **Reproduction :** pour $i = 1$ jusque $m-1$ par pas de 2, remplacer les parents $b_{i,t+1}$ et $b_{i+1,t+1}$ par leurs 2 enfants issus d'un crossover à 1 point de coupure avec une probabilité p_C ;
   - 2.3. **Mutation :** pour $i = 1$ jusque $m-1$, muter $b_{i,t+1}$ par une $1-$inversion avec une probabilité de p_M.

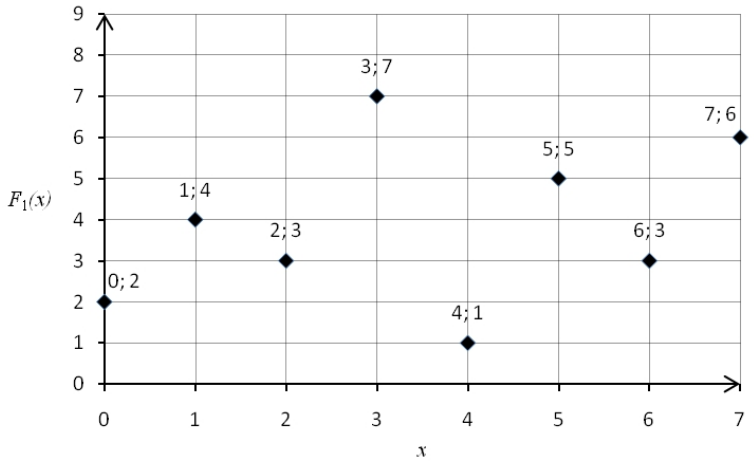


FIGURE 2.5 – Graphe de la fonction discrète F_1.

Considérons maintenant le problème de maximisation où l'ensemble des solutions admissibles est défini par $X_1 := \{0, 1, \ldots, 7\}$ et où la fonction objectif F_1 est donnée sur la figure 2.5. La solution est de manière évidente $x^\star = 3$, mais la simplicité de ce problème nous permet de détailler toutes les étapes de l'exécution de l'algorithme.

La première chose à faire est de définir les fonctions de codage et de décodage sur l'espace de strings choisi. On considère l'espace des strings S comme l'ensemble des triplets binaires, *i.e.* $S = \{0, 1\}^3$, avec comme fonction de codage c la fonction de représentation binaire à 3 bits des éléments de X_1, et la fonction de décodage d son inverse. On applique l'algorithme 2.2 avec $m = 4$, $n = 3$, $p_C = 1$ et $p_M = 0.1$. Simulons et commentons maintenant l'exécution de cet algorithme sur le problème donné.

Pas 1 : Le premier pas de l'algorithme calcule une génération initiale de manière aléatoire :

i	Génotype b_i	Phénotype x	Fitness $f(x)$	$P(b_i$ sélectionné)
1	0 1 0	2	3	0.27
2	0 0 1	1	4	0.36
3	1 0 0	4	1	0.09
4	1 1 0	6	3	0.27

Le total de la fitness est 11 et la moyenne est 2.74. La dernière colonne donne les probabilités de sélection de chaque individu. On voit clairement que les individus 1, 2 et 4 auront de bonnes chances pour être sélectionnés pour la reproduction, alors que l'individu 3 aura plus de chance de mourir.

Pas 2.1 : L'étape suivante est la sélection. Supposons que l'individu 2 est sélectionné deux fois et les individus 1 et 4 une fois chacun.

i	Génotype b_i	Phénotype x	Fitness $f(x)$
1	0 0 1	1	4
2	0 1 0	2	3
3	0 0 1	1	4
4	1 1 0	6	3

Pas 2.2 : Croisons ensuite les individus deux par deux grâce à un crossover à 1 point.

i	Parents	Pt. crossover	Enfants	Phénotype
1	0 | 0 1	1	0 1 0	2
2	0 | 0 0	1	0 0 1	1
3	1 0 | 0	2	0 0 0	0
4	1 1 | 0	2	1 1 1	7

Pas 2.3 : La dernière étape avant d'obtenir la génération suivante est la mutation de certains individus. Supposons que seul l'individu 3 est muté, et que la mutation se fait sur son premier gène. La nouvelle population est alors la suivante :

i	Génotype b_i	Phénotype x	Fitness $f(x)$	$P(b_i$ sélectionné)
1	0 1 0	2	3	0.21
2	0 0 1	1	4	0.29
3	0 0 0	4	1	0.07
4	1 1 1	7	6	0.43

La fitness totale est maintenant de 14, ce qui fait une moyenne de 3.5. La fitness a donc légèrement augmenté en moyenne par rapport à la génération précédente. Comme le maximum n'est pas atteint, on réitère la boucle principale de l'algorithme 2.2.

Pas 2.1 : Supposons que l'individu 4 est sélectionné deux fois, et que les individus 1 et 2 sont chacun repris une fois pour la reproduction.

i	Génotype b_i	Phénotype x	Fitness $f(x)$
1	1 1 1	7	6
2	0 0 1	1	4
3	1 1 1	7	6
4	0 1 0	2	3

Pas 2.2 : On reproduit ensuite à nouveau les individus sélectionnés.

i	Parents	Pt. crossover	Enfants	Phénotype	Fitness
1	1 | 1 1	1	1 1 1	5	5
2	0 | 0 1	1	0 1 1	3	7
3	1 | 1 1	1	1 1 0	2	3
4	0 | 1 0	1	0 1 1	3	7

Pas 2.3 : Supposons pour cette itération qu'aucune mutation n'a lieu. La nouvelle génération est donc donnée par la partie de droite du précédent tableau. On peut voir que le maximum est atteint aux individus 2 et 4. On peut aussi calculer que la fitness totale est de 22 et que la moyenne est de 5.5, ce qui constitue encore une augmentation par rapport à la génération précédente.

Cet exemple basique illustre bien le rôle de la sélection et de la reproduction d'un algorithme génétique. Cependant, dans l'exécution développée, et parce que l'exemple considéré est très basique, la mutation ne joue pas ici un rôle déterminant. Mais pour d'autres exécutions, et en général pour les problèmes "réels", l'opérateur de mutation est absolument *nécessaire* pour ne pas stagner dans un maximum local. Par exemple, pour la population initiale $B_0 = \{(0,1,0),(1,1,0),(0,0,0),(1,0,0)\}$, l'optimum $(0,1,1)$ ne peut être atteint sans mutation puisque l'allèle de son dernier gène n'est présente dans aucun des individus de B_0.

Références pour le chapitre 2

Ce chapitre est basé principalement sur deux sources : d'une part le syllabus du cours [Leclercq, 2010] pour le fonctionnement général de l'algorithme, et d'autre part sur les notes [Bodenhofer, 2002], également pour la description de l'algorithme, mais également pour les opérateurs de sélection, de reproduction et de mutation.

Chapitre 3

Fondements théoriques

Ce chapitre a pour but de donner une justification de l'utilisation des algorithmes génétiques, d'expliquer pourquoi et comment ceux-ci fonctionnent. Il se base principalement sur un résultat : le célèbre "théorème des schémas" de J. Holland (1975). Ce théorème ne constitue en rien une preuve de convergence comme il en existe pour les méthodes d'optimisation traditionnelles, comme par exemple les méthodes de gradient ou de Newton, mais permet d'expliquer pourquoi les GAs amènent à de bons résultats pour un grand nombre de problèmes.

L'énoncé du théorème et sa démonstration sont donnés dans la section 3.1. Nous donnerons ensuite une interprétation de celui-ci dans la section 3.2, avec notamment une illustration sur un exemple simple. Nous discuterons ensuite plusieurs questions soulevées par ce théorème dans la section 3.3.

3.1 Théorème des schémas

Rappelons rapidement le résultat de l'exécution du GA sur le problème simple de la section 2.8. Les individus obtenus à chaque génération, ainsi que leur fitness respective, sont donnés dans le tableau 3.1. Il semblerait d'après ce tableau que le fait de comporter un "0" dans le dernier gène constitue un avantage pour les individus. Nous pouvons remarquer aussi que le nombre d'individus vérifiant cette propriété augmente au cours des générations (il passe de 1 individu à 3 individus dans la population finale). Dans cet exemple basique il semblerait que le GA permette d'augmenter le nombre d'individus ayant des gènes qui influencent positivement sur leur qualité (ici le dernier gène). Le théorème des schémas permet d'affirmer que ce phénomène n'est pas une coïncidence. Avant de présenter le théorème proprement dit, donnons quelques définitions qui s'avèreront utiles dans la suite du chapitre.

Génération 0		Génération 1		Génération 2	
Individus	Fitness	Individus	Fitness	Individus	Fitness
0 1 0	3	0 1 0	3	1 1 1	5
0 0 1	4	0 0 1	4	0 1 1	7
1 0 0	1	0 0 0	1	1 1 0	3
1 1 0	3	1 1 1	6	0 1 1	7

TABLE 3.1 – Tableau récapitulatif de l'exécution du GA sur le problème de la section 2.8.

3.1.1 Définitions préliminaires

Nous nous plaçons dans le cadre d'un algorithme génétique défini sur l'ensemble des strings binaires de longueur n, c'est-à-dire $S = \{0,1\}^n$.

Un string $h := (h_1, \ldots, h_n)$ sur l'alphabet $\{0, 1, *\}$ est appelé *schéma* de longueur n. Les composantes $h_i = *$ constituent les *composantes génériques*[1], alors que les composantes $h_i \neq *$ forment les *spécifications* du schéma. Un schéma peut être vu comme un sous-ensemble de S, et correspond à l'ensemble des strings dont les composantes spécifiées sont fixées. Plus précisément, à un schéma h correspond implicitement l'ensemble

$$s_h := \{s \in S : \ s_i = h_i \ \forall i : h_i \neq *\}. \tag{3.1}$$

Nous dirons qu'un string s dans cet ensemble *satisfait* le schéma h, et on note abusivement $s \in h$.

Nous définissons aussi l'*ordre* O_h d'un schéma h comme le nombre de spécifications qu'il comporte, *i.e.*

$$O_h := |\{i \in \{1, \ldots, n\} : \ h_i \neq *\}|. \tag{3.2}$$

Enfin, la *longueur définissante* δ_h d'un schéma h est la distance entre sa première et sa dernière spécification, *i.e.*

$$\delta_h := \max\{i : \ h_i \neq *\} - \min\{i : \ h_i \neq *\}. \tag{3.3}$$

Par exemple, le schéma $h = (*, *, 1, *, 0, *, *, *)$ est un schéma d'ordre $O_h = 2$ et de longueur définissante $\delta_h = 5 - 3 = 2$. Il y a en tout 2^6 strings de S qui vérifient h (2 possibilités pour chaque composante générique). Par exemple le string $(1, 0, 1, 1, 0, 0, 1, 0)$ vérifie h.

Maintenant ces définitions établies, posons les notations suivantes :

1. Le nombre d'individus d'une population B_t qui satisfont un schéma h est noté $N(B_t \cap h)$.

1. Traduction proposée pour le terme anglais "Wildcard".

2. La moyenne de la fitness sur la population B_t est notée $\overline{f}(B_t)$.
3. La moyenne de la fitness sur l'ensemble des individus de B_t qui satisfont un schéma h est notée $\overline{f}(B_t \cap h)$, et vaut

$$\overline{f}(B_t \cap h) := \frac{1}{N(B_t \cap h)} \sum_{i \in \{j:\ b_{j,t} \in h\}} f(b_{j,t}). \tag{3.4}$$

3.1.2 Théorème

Maintenant ces définitions établies, nous pouvons donner l'énoncé et la preuve du théorème des schémas.

Théorème 3.1 (Théorème des schéma (Holland 1975))**:**
Soit l'algorithme génétique 2.2 défini dans la section 2.8, c'est-à-dire l'algorithme utilisant une sélection par roulette, un crossover à 1 point et une mutation par 1−inversion. Supposons par ailleurs que la fitness est une fonction *déterministe*. Alors, pour tout schéma h, l'inégalité suivante est vérifiée :

$$E\left(N(B_{t+1} \cap h)\right) \geq N(B_t \cap h) \cdot \frac{\overline{f}(B_t \cap h)}{\overline{f}(B_t)} \cdot (1 - p_C \frac{\delta_h}{n-1}) \cdot (1 - p_M \frac{O_h}{n}).$$

Preuve. Soit h un schéma donné tel que $N(B_t \cap h) > 0$. Pour savoir combien d'individus satisferont encore ce schéma à l'étape $t+1$, il faut calculer le nombre d'individus satisfaisant h après les étapes (1) de sélection, (2) de crossover et (3) de mutation.

1. *Nombre d'individus satisfaisant h après la sélection :*
   Rappelons d'abord que chaque individu $b_{i,t}$ a une probabilité d'être sélectionné donnée par

$$P(b_{i,t} \text{ est sélectionné}) = \frac{f(b_{i,t})}{\sum_{j=1}^{m} f(b_{j,t})}. \tag{3.5}$$

   Dès lors, la probabilité qu'a le schéma h d'être sélectionné est donnée par la probabilité qu'au moins un individu de B_t qui le satisfait soit sélectionné, *i.e.*

$$P(h \text{ est sélectionné}) = \frac{\sum_{i \in \{j: b_{j,t} \in h\}} f(b_{i,t})}{\sum_{i=1}^{m} f(b_{i,t})}. \tag{3.6}$$

   Comme on considère une sélection "avec remise", cette probabilité ne change pas dans la boucle de sélection (pas 2.1). De plus, chacun des m

individus est sélectionné indépendamment des autres. Dès lors, le nombre d'individus sélectionnés qui satisfont h suit une loi de probabilité binomiale, dont la probabilité de succès est donnée par l'équation (3.6) et dont le nombre d'épreuves vaut m.

Puisque l'espérance d'une loi binomiale X de paramètres (n_X, p_X) est donnée par $E(X) = n_X p_X$, nous avons que l'espérance du nombre d'individus sélectionnés qui satisfont h vaut

$$\begin{aligned} E\left(N(B_S \cap h)\right) &= m \cdot \frac{\sum_{i \in \{j : b_{j,t} \in h\}} f(b_{i,t})}{\sum_{i=1}^{m} f(b_{i,t})} \\ &= N(B_t \cap h) \cdot \frac{\frac{\sum_{i \in \{j : b_{j,t} \in h\}} f(b_{i,t})}{N(B_t \cap h)}}{\frac{\sum_{i=1}^{m} f(b_{i,t})}{m}} \\ &= N(B_t \cap h) \frac{\overline{f}(B_t \cap h)}{\overline{f}(B_t)}. \end{aligned} \tag{3.7}$$

2. *Nombre d'individus satisfaisant h après la reproduction :*
Lorsque deux individus sont croisés, s'ils satisfont tous les deux h, alors il est clair que leurs deux enfants satisferont encore h. Le nombre d'individus satisfaisant h ne peut donc décroître que si un individu satisfaisant h est croisé avec un individu ne satisfaisant pas h, mais seulement lorsque le point de crossover est choisi entre la première et la dernière spécification de h. Par la formule du nombre de cas probables sur le nombre de cas possibles, la probabilité de choisir un tel point est donné par $\frac{\delta_h}{n-1}$. La probabilité p_s qu'un individu s satisfaisant h, croisé avec n'importe quel autre individu, engendre (au moins) un enfant qui satisfasse encore h après un crossover à 1 point est donc minorée par

$$p_s \geq 1 - p_C \cdot \frac{\delta_h}{n-1}, \tag{3.8}$$

où nous avons multiplié le dernier terme par p_C parce que le croisement n'est effectué qu'avec cette probabilité.

Comme les opérations de sélection et de reproduction se font indépendamment l'une de l'autre dans l'algorithme 2.2, l'espérance du nombre d'individus satisfaisant h après la reproduction est donnée par

$$\begin{aligned} E\left(N(B_R \cap h)\right) &= E\left(N(B_S \cap h)\right) \cdot p_s \\ &\geq N(B_t \cap h) \frac{\overline{f}(B_t \cap h)}{\overline{f}(B_t)} \cdot p_s \\ &= N(B_t \cap h) \frac{\overline{f}(B_t \cap h)}{\overline{f}(B_t)} \left(1 - p_C \frac{\delta_h}{n-1}\right). \end{aligned} \tag{3.9}$$

3. *Nombre d'individus satisfaisant h après la mutation :*
   Enfin, après le crossover le nombre d'individus satisfaisant h ne peut diminuer que si l'un de ces individus est altéré par la mutation à un gène qui correspond à une spécification de h. Comme nous avons considéré la mutation par 1−crossover, la probabilité qu'un individu satisfaisant h soit muté est de p_M, et la probabilité pour que la mutation sur cet individu se fasse sur l'une des spécifications de h est de $\frac{O_h}{n}$. La probabilité pour qu'un individu satisfaisant h ne soit pas altéré par la mutation à l'un des gènes spécifiés est donc de $1 - p_M \frac{O_h}{n}$. De nouveau, comme l'opération de mutation se fait de manière indépendante des deux opérations précédentes, l'espérance du nombre d'individus satisfaisant h après le pas de mutation, et donc dans la génération suivante est minorée par

$$\begin{aligned} E\left(N(B_M \cap h)\right) = E\left(N(B_{t+1} \cap h)\right) &= E\left(N(B_R \cap h)\right) \cdot \left(1 - p_M \frac{O_h}{n}\right) \\ &\geq N(B_t \cap h) \cdot \frac{\overline{f}(B_t \cap h)}{\overline{f}(B_t)} \cdot \left(1 - p_C \frac{\delta_h}{n-1}\right) \cdot \left(1 - p_M \frac{O_h}{n}\right), \end{aligned}$$

   et la thèse est vérifiée. □

Il est possible de généraliser ce résultat pour les autres opérateurs de crossover et de mutation présentés dans les sections 2.5 et 2.6. Nous pouvons énoncer le corollaire suivant :

Corollaire 3.1:
Soit un algorithme génétique de la forme 2.2, où l'opérateur de sélection est défini par la roulette, et où les opérateurs de reproduction et de mutation sont choisis parmi les versions considérées dans les sections 2.5 et 2.6. Alors, pour tout schéma h l'inégalité suivante est vérifiée :

$$E\left(N(B_{t+1} \cap h)\right) \geq N(B_t \cap h) \cdot \frac{\overline{f}(B_t \cap h)}{\overline{f}(B_t)} \cdot P_C(h) \cdot P_M(h), \qquad (3.10)$$

où $P_C(h)$ et $P_M(h)$ sont des constantes qui ne dépendent que du schéma h et des opérateurs de reproduction et de mutation respectivement.

Pour les variantes considérées des opérateurs de reproduction, nous avons en particulier :

1−crossover	$P_C(h) = 1 - p_C \frac{\delta_h}{n-1}$
Crossover uniforme	$P_C(h) = 1 - p_C \left(1 - 0.5^{O_h}\right)$
Tout autre crossover	$P_C(h) = 1 - p_C$

Et pour les opérateurs de mutation considérés, nous avons :

Mutation par 1−Inversion	$P_M(h) = 1 - p_M \frac{O_h}{n}$
Mutation par inversion aléatoire	$P_M(h) = (1 - p_M)^{O_h}$
Inversion totale	$P_M(h) = 1 - p_M$
Sélection aléatoire	$P_M(h) = 1 - p_M \frac{\|h\|}{2^n}$.

Ces résultats peuvent se déduire assez facilement, de manière analogue à ce qui est fait dans la preuve du théorème des schémas pour les opérateurs de 1−crossover et de 1−inversion.

3.2 Interprétation

Cette section a pour but d'expliquer comment le théorème des schémas permet de justifier le bon fonctionnement des GAs pour un grand nombre de problèmes.

Le théorème des schémas donne en fait une information par rapport à l'évolution des *schémas* au cours de l'exécution de l'algorithme. Il exprime le fait que l'algorithme génétique va privilégier et augmenter le nombre de schémas avec une fitness au-dessus de la moyenne (les "bons individus") et dont la longueur définissante est petite. Dans la démonstration, nous pouvons voir que la première propriété est due à l'opérateur de sélection, et que la seconde est due à l'opérateur de crossover, dans le sens où celui-ci détruit plus facilement les schémas dont la longueur définissante est grande. De tels schémas vont être appelés par la suite des *blocs élémentaires* [2].

Le fonctionnement général d'un GA est donc de détecter d'une certaine manière les gènes, ou les petits groupes de gènes, qui influencent positivement la qualité des individus, et de les avantager dans les générations futures. Évidemment, le GA manipule un ensemble de strings et non un ensemble de schémas, et donc ce mécanisme se fait de manière implicite. De plus, comme un même string peut satisfaire un très grand nombre de schémas différents, le nombre total de schémas différents manipulés simultanément par le GA est aussi très grand. Cette caractéristique est appelée le *parallélisme implicite* des GAs.

Donnons maintenant une idée du nombre de schémas manipulés dans une génération de m individus. Calculons d'abord le nombre de schémas que satisfait un string de taille n. Le nombre de schémas d'ordre k que satisfait un string de taille n est le nombre de possibilité de choisir k composantes

2. Traduction proposée pour le terme anglais "building blocks".

spécifiées parmi n, et vaut donc C_k^n. En sommant sur tous les ordres de schémas possibles, nous obtenons qu'un string satisfait au total $\sum_{i=1}^{n} C_k^n = 2^n$ schémas. Comme une population de taille m contient entre 1 et m individus différents, le nombre de schémas varie donc entre 2^n et $m2^n$.

Pour illustrer le résultat donné par le théorème des schémas sur un exemple concret, reprenons l'exemple présenté au début de ce chapitre. Nous avions remarqué que lorsque l'allèle du dernier gène avait la valeur 1, cela semblait constituer un avantage, une garantie de qualité pour les individus. Nous allons vérifier l'inégalité donnée par le théorème des schémas pour le schéma $h = (*, *, 1)$. Rappelons tout d'abord que nous avions posé les paramètres suivants pour l'algorithme : $p_C = 1$ et $p_M = 0$. Ensuite, on vérifie facilement que ce schéma h est d'ordre 1 et a une longueur définissante nulle. Il est clair également que la fitness moyenne $\overline{f}(B_0 \cap h)$ pour ce schéma dans la population initiale vaut 4, la valeur de l'unique individu $b = (0, 0, 1)$ satisfaisant h, et que la moyenne globale $\overline{f}(B_t)$ est quant à elle égale à 2.75. Le théorème des schémas implique donc que

$$E(N(B_1 \cap h)) \geq 1 \cdot \frac{4}{2.75}(1 - 0.9 \cdot 0)(1 - 0 \cdot \frac{1}{3}) \simeq 1.45. \tag{3.11}$$

On peut donc espérer avoir au moins 1.45 individu(s) dans B_1 satisfaisant h. Ce résultat est vérifié puisque le nombre d'individus satisfaisant h dans la population suivante est de 2. On peut procéder de la même manière pour calculer le nombre espéré d'individus satisfaisant h dans la 2^e génération, on trouve

$$E(N(B_2 \cap h)) \geq 2 \cdot \frac{5}{3.25}(1 - 0.9 \cdot 0)(1 - 0 \cdot \frac{1}{3}) \simeq 3.08. \tag{3.12}$$

De nouveau, ce résultat est vérifié car la 2^e génération contient 3 individus satisfaisant le schéma h.

Le mécanisme des GAs semble assez puissant, puisqu'il permet de détecter les facteurs influençant la qualité des individus. Dans notre exemple basique, le GA a permis de proliférer la caractéristique "contenir un dernier gène de valeur 1", qui correspond en fait au niveau de l'ensemble des solutions admissibles X à mettre en avant les candidats *impairs*.

Cependant, la caractéristique d'augmenter le nombre de schémas dont la fitness moyenne est élevée et dont la longueur définissante est petite ne garantit en rien la convergence de la méthode, et plusieurs questions peuvent être posées quant à la consistance de cette stratégie. Nous tentons de répondre à ces questions dans la section suivante.

3.3 Discussion

3.3.1 Exploration et exploitation

Tout d'abord, nous avons vu qu'une conséquence du théorème des schémas était l'augmentation d'une génération à l'autre des schémas dont la fitness est au-dessus de la moyenne et dont la longueur est assez petite. En reprenant l'inégalité (3.10) donnée par le théorème des schémas, il est clair que ce nombre va augmenter tant que la fitness du schéma considéré est plus grande que la fitness moyenne observée. Supposons être en présence d'un tel schéma. L'augmentation du nombre d'individus satisfaisant ce schéma va correspondre en fait à la propagation des gènes spécifiés du schéma auprès des individus de la population, augmentant d'une part la qualité globale de la population, mais également l'homogénéité de la population relativement à ces gènes. L'algorithme va donc diminuer progressivement l'*exploration* de nouveaux allèles pour les gènes concernés à l'avantage de l'*exploitation* de l'information qu'il possède déjà par rapport à ces gènes. Il est donc légitime de se demander si cette stratégie est une bonne stratégie, et si oui, pourquoi c'est le cas.

Certains justifient cette stratégie par une comparaison avec un problème de statistique décisionnelle : le problème de la "machine à sous à deux leviers" [3]. Supposons être en possession d'une machine à sous qui possède deux fentes pour insérer des pièces de monnaie, ainsi que deux leviers, un pour chaque fente. Lorsque le joueur actionne l'un des leviers, il peut soit gagner une récompense, soit perdre la pièce de monnaie insérée. Chaque levier est caractérisé par un certain gain moyen μ_i et une variance σ_i^2, inconnus par le joueur. Le joueur joue N fois d'affilée en choisissant à chaque étape un levier à actionner. Il doit donc prendre une suite de décisions concernant le levier à actionner à chaque moment dans le but d'avoir le gain le plus grand possible, en fonction des réalisations précédentes. Le joueur doit donc d'une part varier suffisamment ses choix pour avoir le plus d'informations possibles sur le gain respectif des deux leviers, mais il doit aussi d'autre part utiliser l'information qu'il a récoltée pour tenter d'avoir un gain maximum. On retrouve donc de nouveau un dilemme entre exploration et exploitation de l'information.

L'approche la plus simple est sans doute de séparer l'exploration de l'exploitation, et de commencer par tester un certain nombre de fois chaque levier, pour ensuite exploiter l'information obtenue en choisissant pour le reste des parties le levier qui possède le gain moyen le plus élevé. Ce problème a été étudié par [Holland, 1992], et il a été montré que la stratégie

3. Traduction proposée pour le terme anglais "The two-armed bandit problem".

optimale pour ce problème consistait à consacrer d'abord

$$n^* \simeq (\frac{\sigma_1}{\mu_1 - \mu 2})^2 \ln \left(\frac{N^2}{8\pi b^4 \ln(N^2)} \right) \tag{3.13}$$

essais pour chaque levier, et d'ensuite choisir le levier qui a rapporté le gain moyen de le plus élevé pour le reste du jeu. Le joueur actionnera donc au total n^* fois le levier dont le gain moyen observé est le plus faible et $N - n^*$ fois l'autre levier. Une analyse rapide permet de voir que la proportion $\frac{n^*}{N}$ est une fonction strictement décroissante en fonction de N et tend vers 0, alors que la proportion $\frac{N-n^*}{N}$ est croissante et tend vers 1. Plus précisément, si on trace la courbe d'évolution des points $(n^*, N - n^*)$ en fonction de N, il apparaît que le nombre d'essais alloués au meilleur levier observé croît de manière exponentielle par rapport au nombre d'essais alloués à l'autre bras. Cela signifie que plus le nombre de parties est grand, plus la phase d'exploration est petite par rapport à la phase d'exploitation.

Ce résultat permet de justifier le comportement du GA qui augmente progressivement le nombre de schémas dont la fitness observée est au-dessus de la moyenne. Pour faire le lien avec le problème de la machine à sous à deux leviers, remarquons que pour chaque gène pris individuellement, il y a deux possibilités d'allèles et donc deux schémas d'ordre 1 qui spécifient ce gène. Le GA est donc confronté à faire un choix à chaque étape entre les deux schémas, à partir de l'information limitée de la moyenne de la fitness pour chaque schéma. Autrement dit, celui-ci doit résoudre un grand nombre de problèmes de machines à sous à deux leviers en parallèle. Cependant le théorème des schémas ne se limite pas aux seuls schémas d'ordre 1, mais aux schémas de tout ordre. Le GA doit donc résoudre à chaque étape un grand nombre de problèmes de machines à sous à k leviers en parallèle. Bien que plus complexe à résoudre, la généralisation du problème à k leviers peut être résolue de manière analogue ([Holland, 1992]) et la solution optimale consiste encore à donner un nombre d'essais de plus en plus grand à la meilleure alternative observée en fonction du nombre d'essais total. Transposé au problème des GAs, cela signifie que la meilleure stratégie pour obtenir une fitness totale maximale consiste bien à privilégier progressivement les schémas dont la fitness observée est au-dessus de la moyenne et de préférer au cours des générations l'exploitation à l'exploration.

3.3.2 Hypothèse des blocs élémentaires

Nous avons vu jusqu'à présent que le théorème des schémas permettait de privilégier les schémas de petite taille dont la fitness est au-dessus de la moyenne. Cependant, rien ne garantit à ce stade une bonne performance de l'algorithme, c'est-à-dire sa convergence vers une solution optimale. Comme nous l'avons déjà mentionné, la convergence des GAs ne peut être garantie

de la même manière que les méthodes conventionnelles de gradient ou de Newton, même si certains travaux donnent des pistes dans ce domaine, citons par exemple la thèse [Cerf, 1994].

Le fonctionnement de la méthode repose principalement sur une hypothèse, appelée l'*hypothèse des blocs élémentaires*[4], et qui consiste à affirmer qu'"un algorithme génétique permet de créer des solutions meilleures à chaque itération par l'application des opérateurs de reproduction et de mutation sur les schémas de petite taille et de haute fitness".

Nous avions déjà fait allusion à cette hypothèse dans le chapitre 2 lors de l'introduction des opérateurs de reproduction, en motivant leur utilisation par le fait que l'espoir était d'obtenir d'aussi bons enfants, voire potentiellement meilleurs, que les parents desquels ils sont issus. L'hypothèse consiste en fait à affirmer que les individus optimaux pour le problème considéré peuvent être atteints par une suite de reproductions et de mutations d'individus possédant des schémas de petite taille définissante et de haute fitness. Cela revient en quelque sorte à supposer que la qualité globale de la solution peut être subdivisée en plusieurs contributions locales, qui correspondent aux blocs élémentaires.

Il reste à savoir si cette hypothèse est réaliste ou non. Pour répondre à cette question, considérons deux problèmes particuliers sur $S = \{0,1\}^n$, où les fonctions de fitness f_1 et f_2 sont décrites respectivement par

$$f_1(s) = \sum_{i=1}^{n} c_i s_i, \tag{3.14}$$

où c_i sont des coefficients réels fixés, et par

$$f_2(s) = I_{\{s_0\}}(s), \tag{3.15}$$

où $I_{\{s_0\}}(.)$ désigne la fonction indicatrice en s_0, *i.e.* la fonction nulle partout sauf au point s_0 où elle vaut 1. Il est clair que le premier problème peut être résolu gène par gène, en déterminant $s_i = 1$ lorsque $c_i > 0$ et $s_i = 0$ sinon. Dans ce cas, l'hypothèse est vérifiée, et les blocs élémentaires correspondent aux schémas d'ordre 1. Par contre, pour le deuxième problème, seule la combinaison particulière de tous les gènes influence la qualité de la solution, puisque la fitness est nulle partout sauf en 1 point et qu'aucune information partielle n'est exploitable pour guider la recherche. L'hypothèse n'est donc pas vérifiée dans ce second cas.

De manière générale, la recherche du GA sera plus facile lorsque la qualité peut être jugée sur des sous-groupes de gènes de manière indépendante, et

4. Terme anglais : "Building Block Hypothesis".

sera au contraire plus difficile lorsque la qualité dépend de manière couplée de plusieurs groupes de gènes. Remarquons cependant que cette caractéristique n'est pas propre uniquement aux GAs, mais de manière générale à l'ensemble des méthodes d'optimisation, qui sont généralement moins efficaces sur les problèmes hautement non-linéaires. Un terme biologique existe pour traduire ce genre d'interactions entre plusieurs gènes : l'*épistasie*. Il semblerait que les GAs soient appropriés pour résoudre des problèmes d'épistasie moyenne, les méthodes plus classiques étant plus performantes pour les problèmes à faible épistasie, alors qu'il n'y a pas vraiment de meilleure méthode pour les problèmes à haute épistasie.

Plaçons nous maintenant au niveau des strings manipulés par le GAs, en regardant si deux parents de bonne qualité engendrent toujours des enfants de qualité comparable (ou meilleure). La réponse est négative, et pour s'en convaincre nous allons considérer un exemple basique sur l'ensemble $\{0,1,2,3\}$ représenté par l'ensemble des strings binaires de longueur 2. Le codage est choisi simplement comme la représentation binaire naturelle. Considérons une fonction objectif f définie par $f(0,0)=6$, $f(0,1)=5$, $f(1,0)=1$ et $f(1,1)=7$. La solution globale est donnée par le point $(1,1)$. Cependant, dans cet exemple basique, les deux meilleurs parents engendrent les deux individus les moins bons, et inversement. On peut également calculer que le schéma d'ordre 1 $(0,*)$, qui correspond à un schéma de solutions sous-optimales, est meilleur que le schéma d'ordre 1 $(1,*)$ que vérifie la solution optimale. En effet :

$$\overline{f}(0,*) = 5.5 > 4 = \overline{f}(1,*). \tag{3.16}$$

Le schéma qui sera privilégié par le mécanisme des GAs sera donc le schéma $(0,*)$, qui ne mène pas à l'optimum.

Cet exemple basique illustre bien que l'hypothèse sur laquelle repose les GAs n'est pas toujours vérifiée. Cependant, cela ne signifie pas pour autant que le GA ne sera pas capable d'atteindre l'optimum, car la reproduction se fait également sur des individus moins adaptés, et cela peut mener, comme dans l'exemple précédent, à de bien meilleurs individus. Cependant, lorsque l'épistasie n'est pas trop élevée, le choix du codage peut avoir une forte influence dans les performances du GA.

3.3.3 Importance du codage

Dans cette section nous mettons en avant l'importance du choix de la fonction de codage. Pour ce faire, reprenons d'abord l'exemple précédent. Il est possible de définir pour l'exemple précédent un codage adapté qui évite les problèmes rencontrés. Par exemple, en considérant le codage c défini par

$$c(0) := (0,1),\ c(1) := (1,0),\ c(2) := (1,1) \text{ et } c(3) := (0,0), \tag{3.17}$$

le problème est évité car la solution optimale est représentée par $(0,0)$, et que les schémas sous-optimaux ont une fitness plus petite que les schémas que vérifie la solution optimale. En effet, on peut calculer que

$$\overline{f}(0,*) = 7.5,\ \overline{f}(1,*) = 3,\ \overline{f}(*,0) = 6 \text{ et } \overline{f}(*,1) = 5. \tag{3.18}$$

Dès lors, les schémas qui seront privilégiés sont les schémas $(0,*)$ et $(*,0)$, qui mènent à l'optimum. On peut aussi vérifier que le croisement des deux meilleurs individus $(0,1)$ et (00) engendre des enfants identiques, et que le croisement du deuxième et troisième meilleurs individus engendre (notamment) l'optimum. Dans ce cas, l'hypothèse des blocs élémentaires est vérifiée. Plus particulièrement, la qualité de l'individu est déterminée avec ce codage par le fait de "comporter le plus de 0 possible".

Le codage doit permettre d'une certaine manière de regrouper les meilleurs individus selon certains critères faisant intervenir les gènes. Autrement dit, il faut idéalement que les meilleurs individus partagent certains schémas. Pour notre exemple, il suffit de choisir une représentation où l'individu 0, 1 et 3 partagent un même gène, et où les individus 2 et 3 n'ont aucun gène en commun.

Cependant, en pratique on connait rarement la structure de la fonction objectif, et il n'est donc pas possible de savoir a priori si le codage sera adapté. Cependant, certains codages sont connus pour être mieux adaptés que d'autres pour certaines classes de problèmes. Par exemple, lorsque l'ensemble des solutions admissibles est un sous-ensemble de $\mathbb{Z}$, la représentation binaire signée n'est pas idéale si la fonction objectif prend des valeurs qui ne varient pas beaucoup autour de son optimum supposé être en 0. En effet, alors les nombres -1 et 0, successifs dans $\mathbb{Z}$, seront totalement différents dans leur représentation codée puisqu'ils seront représentés respectivement par $(1,1,\ldots,1)$ et $(0,0,\ldots,0)$. Dès lors, le GA convergera vers -1 ou 0 suivant la proportion d'éléments positifs et négatifs dans la population initale. Pour ce type de problème, la *représentation de Grey* est souvent préférée, car elle permet de garantir que deux chiffres successifs soient différents seulement d'un bit.

Le codage de Grey g peut être défini entre $X_n = \{0,1,\ldots,2^n-1\}$ et $S_n = \{0,1\}^n$ par récurrence sur n de la manière suivante :

- si $n = 1$, alors la relation est définie par $g_1(0) = 0$ et $g_1(1) = 1$;
- supposons la relation définie pour n par

$$g_n(0) = s_0, g_n(1) = s_1, \ldots, g_n(2^n-1) = s_{2^n-1}, \tag{3.19}$$

où $s_i \in \{0,1\}^n$ pour tout $i\{0,\ldots,2^n-1\}$.

La relation pour $n+1$ est définie $\forall 0 \leq k \leq 2^{n+1}-1$ par

$$g_{n+1}(k) = \begin{cases} (0, [s_k]_1, \ldots, [s_k]_{2^n-1}) & \text{si } k \leq 2^n - 1 \\ (1, [s_{2^{n+1}-1-k}]_1, \ldots, [s_{2^{n+1}-1-k}]_{2^n-1}) & \text{sinon.} \end{cases} \quad (3.20)$$

On obtient par exemple les correspondances suivantes pour $n = 1$, $n = 2$ et $n = 3$:

- $n = 1 : 0 \leftrightarrow 0$ et $1 \leftrightarrow 1$.
- $n = 2 : 0 \leftrightarrow 00$, $1 \leftrightarrow 01$, $2 \leftrightarrow 11$ et $3 \leftrightarrow 10$.
- $n = 3 : 0 \leftrightarrow 000$, $1 \leftrightarrow 001$, $2 \leftrightarrow 011$, $3 \leftrightarrow 010$, $4 \leftrightarrow 110$, $5 \leftrightarrow 111$, $6 \leftrightarrow 101$ et $7 \leftrightarrow 100$.

Notons qu'un codage différent peut se faire au niveau du choix de la fonction de codage comme pour la représentation de Grey, mais également dans le choix de l'ensemble des strings S. Par exemple, on peut être confronté à choisir entre une représentation de type binaire ou de type décimale. La différence entre les deux représentations se situe dans le nombre de gènes par chromosome et le nombre d'allèles par gène. Il est généralement fait le choix de préférer les représentations dont le nombre de schémas manipulés par l'algorithme est plus grand, car cela permet un plus grand brassage des individus lors des opérations de variation.

3.4 Conclusion

Ce chapitre a permis de donner un aperçu des fondements sur lesquels sont basés les GAs. Rappelons brièvement les résultats présentés.

Tout d'abord nous avons vu, grâce au théorème des schémas, que le nombre de schémas de petite taille et de fitness au-dessus de la moyenne augmente progressivement au cours des générations. Nous avons donné une justification de cette stratégie grâce au problème des machines à sous à leviers multiples.

Ensuite, nous avons vu que malgré le fait que les GAs manipulent un ensemble de m individus, ceux-ci traitent un nombre beaucoup plus grands de schémas de manière implicite.

Enfin, nous avons donné l'hypothèse principale sur laquelle repose le fonctionnement des GAs, et nous avons vu que celle-ci n'est pas forcément toujours vérifiée. Nous avons aussi signalé que la représentation des individus peut jouer un rôle important dans le respect de cette hypothèse.

Références pour le chapitre 3

Ce chapitre s'inspire des deux mêmes sources que le chapitre précédent, pour rappel [Leclercq, 2010] et [Bodenhofer, 2002]. Plus particulièrement, nous nous sommes inspirés de la première source pour la section 3.3.3 relative à l'importance de la représentation, et de la deuxième pour la présentation du théorème des schémas et des discussions de la section 3.3. Seules les illustrations sur des exemples concrets et la section 3.2 relative à l'interprétation du théorème des schémas sont un peu plus personnelles.

Chapitre 4

Robby, the Soda-Can-Collecting Robot

Ce chapitre est consacré à l'application des algorithmes génétiques sur un problème plus concret et plus conséquent que le problème basique utilisé dans le chapitre 1, celui de la recherche de stratégie optimale pour un robot, "Robby", dont la tâche consiste à nettoyer une surface donnée. *Robby, the Soda-Can-Collecting Robot* est un problème qui a été posé comme exercice dans le cours *CS 441/541 Artificial Intelligence* de Mélanie Mitchell, professeur à l'université de Portland. La présentation que nous faisons ici se base sur son livre [Mitchell, 2009].

4.1 Présentation du problème

Robby est un robot dont le but est de nettoyer une surface en collectant les cadavres de canettes qui y sont abandonnées. Une telle surface est représentée par une grille de 10×10 cases, dont certaines contiennent une canette de soda alors que d'autres sont vides. Au total, 50 canettes sont déposées dans les cases de manière aléatoire. Robby peut se déplacer comme il veut à l'intérieur de la surface et les frontières de la grille sont assimilées à des murs qu'il ne peut traverser. L'illustration d'une telle surface est faite sur la figure 4.1.

Robby n'est cependant pas un robot très performant car lorsqu'il se situe dans une case, ses capteurs ne lui permettent que de connaître l'état des cases qui lui sont adjacentes dans les directions cardinales, ainsi que l'état de la case sur laquelle il se trouve. Par exemple, sur la figure 4.1, Robby se trouve sur la case $(0, 0)$ et voit qu'au nord et à l'ouest il y a des murs, qu'à l'est il y a une canette, et qu'au sud et sur sa case il n'y a rien. À partir de cette information limitée, il devra alors prendre une décision parmi l'une des 7 possibilités suivantes : se déplacer d'une case vers le nord, vers le sud,

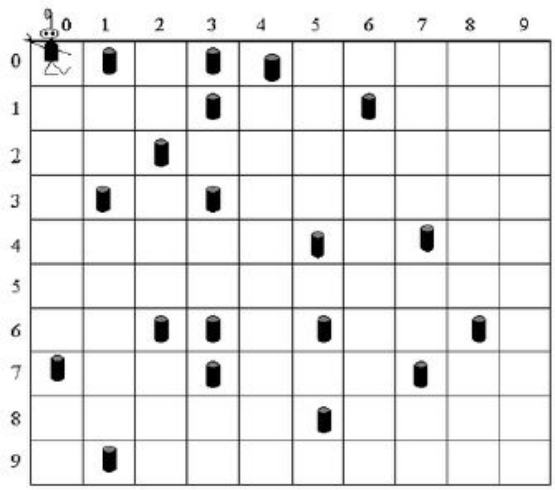


FIGURE 4.1 – Représentation d'une grille. Source : [Mitchell, 2009].

vers l'ouest ou vers l'est, choisir l'une des 4 possibilités précitées de manière aléatoire, ne rien faire, ou s'abaisser et attraper la canette. Après avoir effectué l'une de ces actions, il devra réévaluer son voisinage direct, à nouveau effectuer une action à partir de celui-ci, et ainsi de suite. La définition du comportement de Robby se fait donc en spécifiant l'action qu'il doit effectuer pour chacun de ses états possibles.

Lors de chaque "session de nettoyage", Robby a droit à exactement 200 actions, en partant de la case supérieure gauche. Son but est donc de ramasser le plus de canettes possibles endéans ce nombre limité d'actions, et en évitant par exemple de se prendre des murs ou de tenter de ramasser des canettes dans les cases où il n'y a rien.

L'idée pour déterminer le comportement amenant au meilleur nettoyage est d'appliquer les algorithmes génétiques sur l'ensemble des stratégies de nettoyages possibles.

4.2 Modélisation

Pour pouvoir appliquer un algorithme génétique sur ce problème, il faut d'abord modéliser celui-ci et exprimer de manière mathématique l'ensemble des solutions admissibles X et la fonction objectif à maximiser f.

4.2.1 Ensemble des solutions admissibles

L'ensemble des solutions admissibles correspond à l'ensemble des stratégies différentes pouvant être adoptées par un robot collecteur. Une stratégie peut être définie par une table qui spécifie l'action à effectuer pour chacune des situations qui pourraient être rencontrées. Le nombre des situations possibles est de $3^5 = 243$, puisqu'à chaque instant, le robot connaît l'état de 5 cases (nord, sud, ouest, est et sa case courante) et que chacune d'entre elles a 3 états possibles (est un mur, est vide ou contient une canette). Le tableau 4.1 montre un exemple de stratégie [1]. Cet exemple correspond à un robot qui décide d'aller au nord lorsque ses cases voisines et sa case courante sont vides, d'aller au sud lorsque ses cases voisines sont vides mais que sa case courante contient une canette, etc.

No.	État					Action
	Courante	Nord	Sud	Est	Ouest	
1	vide	vide	vide	vide	vide	Aller au nord
2	vide	vide	vide	vide	can.	Aller au sud
3	vide	vide	vide	vide	mur	Prendre canette
4	vide	vide	vide	can.	vide	Aller en est
⋮	⋮	⋮	⋮	⋮	⋮	⋮
84	mur	vide	vide	can.	can.	Prendre canette
⋮	⋮	⋮	⋮	⋮	⋮	⋮
243	mur	mur	mur	mur	mur	Mouvement aléatoire

TABLE 4.1 – Représentation du codage d'une stratégie de nettoyage.

Évidemment certaines situations sont impossibles, par exemple lorsque plus de deux cases voisines sont des murs, ou lorsque la position courante est un mur. Pour éliminer une partie des états impossibles en pratique, le dernier tiers du tableau est "coupé", de telle manière à ce que tous les états où la case courante est un mur soient ignorés. Le nombre d'états considérés est donc réduit à $243 - \frac{1}{3} \cdot 243 = 162$ possibilités. Cependant, il reste encore des états impossibles dans ces 162 états, mais nous ne filtrons pas *tous* les états impossibles, car il faudrait alors adopter une représentation plus complexe que la représentation naturelle présentée dans la section 4.3. De plus, au niveau du GA cela n'aura pas un grand impact puisque ces situations ne sont jamais rencontrées. En résumé, nous manipulerons donc pour chaque stratégie un ensemble de 162 états/actions, qui correspondent au deux premiers tiers du tableau 4.1, même si certaines de ces situations ne seront jamais rencontrées.

1. La stratégie donnée en exemple n'est évidemment pas complète, mais il serait trop long de donner le tableau entier.

Code	Action
A	Aller au nord
B	Aller au sud
C	Aller à l'est
D	Aller à l'ouest
E	Mouvement aléatoire
F	Rester sur place
G	Prendre la canette

(a)

Code	État
0	Est vide
1	Contient une canette
2	Est un mur

(b)

TABLE 4.2 – Codage (a) des actions et (b) des états possibles des cases.

En prenant les conventions données par les tableaux 4.2(a) et 4.2(b) pour les actions et les états de chaque case respectivement, une stratégie peut donc être représentée par une fonction discrète x qui associe à chaque état une action que le robot devra effectuer :

$$\begin{array}{rccc} x : & \{0,1,2\}^5 & \longrightarrow & \{A,B,C,\ldots,G\} \\ & (e_1,e_2,e_3,e_4,e_5) & \longmapsto & x(e_1,e_2,e_3,e_4,e_5) \end{array} \tag{4.1}$$

où e_1, e_2, e_3, e_4 et e_5 représentent respectivement l'état de la case courante, nord, sud, est et ouest, et où $x(e_1,e_2,e_3,e_4,e_5)$ désigne l'action à effectuer pour la situation correspondante. Dès lors, l'ensemble des solutions admissibles est donné par l'ensemble de ces fonctions :

$$X = \left\{x : \{0,1,2\}^5 \longrightarrow \{A,B,C,\ldots,G\}\right\}. \tag{4.2}$$

4.2.2 Fonction objectif

La fonction objectif doit refléter la qualité des solutions par rapport à un certain but donné. Rappelons que les stratégies optimales sont définies comme les stratégies permettant de nettoyer "au mieux" n'importe quelle surface donnée en 200 actions maximum.

La définition la plus intuitive pour la fonction objectif est la fonction qui compte le nombre de canettes ramassées lors d'une session de nettoyage. Comme la surface à nettoyer est déterminée de manière aléatoire, on peut définir la fitness comme la moyenne du nombre de canettes ramassées sur n_s sessions, avec n_s assez grand.

Cependant, cette définition ne permet pas de départager les stratégies qui ramassent un même nombre de canettes. Or, il semble assez légitime de préférer une stratégie qui évite les murs, ou qui ne tente pas de ramasser

des canettes dans des cases qui sont vides. Une manière de prendre ces différences en compte est d'ajouter à la fonction objectif un terme de pénalité pour les actions non désirées. En pratique, il y a plusieurs manières de définir ce terme de pénalité, amenant sans doute à des performances et des résultats différents. La définition reprise dans [Mitchell, 2009], et adoptée dans ce travail, consiste à associer

- une récompense de 10 points lorsqu'une canette est ramassée ;
- une pénalité de -5 points lorsque le robot entre en collision avec un mur ;
- une pénalité de -1 point lorsque le robot essaye de ramasser une canette alors qu'il n'y a rien.

La fonction objectif est alors calculée comme étant la moyenne sur n_s sessions de nettoyage du gain accumulé lors de chaque session. Nous avons choisi de fixer $n_s = 100$ pour l'ensemble de nos simulations. Notons $\tilde{f}$ cette définition de la fonction objectif.

Nous choisissons comme fonction objectif la version normalisée dans $[0, 1]$ de $\tilde{f}$. Comme, par construction, cette dernière ne peut prendre que des valeurs entre $\tilde{f}_{\min} = 200 \cdot (-5) = -1000$ (lorsque le robot fonce continuellement dans un mur) et $\tilde{f}_{\max} = 50 \cdot 10 = 500$ (lorsque le robot ramasse toutes les canettes), la fonction objectif considérée dans ce travail est définie par

$$\begin{array}{rccl} f: & X & \longrightarrow & [0,1] \\ & x & \longmapsto & f(x) := \dfrac{\tilde{f}(x) - \tilde{f}_{\min}}{\tilde{f}_{\max} - \tilde{f}_{\min}} \end{array} \tag{4.3}$$

Une telle définition permet d'avoir une fonction objectif qui a une forme adéquate pour la sélection par roulette (cf. section 2.4).

4.3 Algorithme génétique

Maintenant le problème modélisé, donnons la spécification des différents composants de l'algorithme génétique. Il faut définir les fonctions de codage et de décodage, ainsi que donner la structure algorithmique utilisée, avec le choix des opérateurs de sélection, de reproduction et de mutation (cf. chapitre 2).

4.3.1 Fonctions de codage et de décodage

Définissons les fonctions de codage et de décodage sur l'ensemble X. Le choix que nous avons fait est de considérer pour chaque stratégie x le vecteur à $n = 162$ composantes formé des choix d'action pour chaque situation possible, dans l'ordre donné par le tableau 4.1. Autrement dit, chaque stratégie

sera codée grâce à la donnée de la *dernière colonne* de son tableau. L'ensemble des strings S sera donc donné par $S := \{A, B, C, \ldots, G\}^n$.

De manière formelle, la fonction de codage c est définie par

$$\begin{array}{rccc} c: & X & \longrightarrow & S \\ & x & \longmapsto & c(x) \end{array} \tag{4.4}$$

où chaque composante $[c(x)]_i$ du vecteur $c(x)$ est définie par

$$[c(x)]_i = x(a_4(i), a_3(i), a_2(i), a_1(i), a_0(i)), \tag{4.5}$$

et où les nombres $a_j(i)$ sont les coefficients de décomposition en base 3 du nombre i, c'est-à-dire sont les nombres tels que

$$i = \sum_{j=0}^{4} a_j(i) \cdot 3^j \text{ et } a_j \in \{0, 1, 2\}. \tag{4.6}$$

Ces nombres correspondent en fait à l'état des cases courante, nord, sud, est et ouest de la i^e ligne dans la tableau définissant la stratégie.

La fonction de décodage est simplement l'inverse de la fonction de codage, c'est-à-dire la fonction $d : S \longrightarrow X$ telle que $\forall s \in S$, la fonction $d(s) \in X$ est définie par

$$\begin{array}{rccc} d(s): & \{0,1,2\}^5 & \longrightarrow & \{A, B, C, \ldots, G\} \\ & (e_1, e_2, e_3, e_4, e_5) & \longmapsto & d_s(e_1, e_2, e_3, e_4, e_5) := s_{\sum_{j=1}^{5} e_j \cdot 3^{5-j}}. \end{array} \tag{4.7}$$

Remarquons que la définition (4.6) du codage en base 3 est défini normalement pour $n = 243$, c'est-à-dire l'entièreté du tableau définissant la stratégie. Cependant, ne considérer que les 2 premiers tiers du tableau n'aura pour conséquence que le fait d'imposer la condition $i \leq 162$. En base 3, cette condition est équivalente à imposer $a_4 \neq 2$, autrement dit interdire que la case courante soit un mur.

4.3.2 Structure algorithmique

Donnons maintenant la spécification de l'algorithme 2.1 qui a été choisie pour résoudre ce problème.

Signalons d'abord que nous avons implémenté plusieurs versions pour les opérateurs de reproduction, de mutation, et de sélection, de manière à voir si certains opérateurs amènent à de meilleures performances du GA que d'autres. Le tableau 4.3 reprend les différents opérateurs implémentés. Notons que les définitions de ces opérateurs sont données dans le chapitre 2.

Type d'opérateur	Code	Description	Paramètres
Reproduction	R_1	crossover à 1 point	P_C
	R_2	crossover à 2 points	P_C
	R_3	crossover uniforme	P_C
Mutation	M_1	mutation par 1-inversion	P_M
	M_2	inversion aléatoire	P_M
Sélection	S_1	sélection par roulette	/
	S_2	sélection par rang	/
	S_3	sélection par tournoi	p_T

TABLE 4.3 – Tableau reprenant l'ensemble des opérateurs implémentés pour la reproduction, la mutation et la sélection.

Étant choisis des opérateurs de reproduction, de mutation et de sélection dans le tableau précédent, la structure choisie pour le GA est donné par l'algorithme 4.1.

Algorithme 4.1:
Soient donnés les paramètres $m, t_{\max} \in \mathbb{N}_0$, avec m pair. Soient aussi les identifiants R, M et S des opérateurs de reproduction, de mutation et de sélection respectivement, ainsi que leurs paramètres p_C, p_M et (éventuellement) p_T.

1. **Initialisation :** Créer une population initiale $B_0 = (b_{1,0}, \ldots, b_{m,0})$ de taille m, avec des individus choisis aléatoirement dans S, et calculer leur fitness F ;
2. **Corps de l'algorithme :** Pour $t = 0$ jusque t_{max} :
   - 2.1. **Sélection :** sélectionner m individus $(b_{1,t+1}, \ldots, b_{m,t+1})$ dans B_t par la stratégie de sélection S (avec probabilité p_T si S=3) ;
   - 2.2. **Reproduction :** pour $i = 1$ jusque $m - 1$ par pas de 2, remplacer les parents $b_{i,t+1}$ et $b_{i+1,t+1}$ par leurs 2 enfants issus de l'opérateur de reproduction R si rand$(0,1)<p_C$, les garder sinon ;
   - 2.3. **Mutation :** pour $i = 1$ jusque $m-1$, muter $b_{i,t+1}$ par l'opérateur de mutation M si rand$(0,1)<p_M$, le garder sinon ;
   - 2.4. **Calcul de la fitness :** Calculer la fitness F pour chaque individu de la population ;
   - 2.5. **Élitisme :** remplacer $b_{l,t+1}$ par $b_{l,t} = \text{argmax}_{b \in B_t} F(b)$ si $F(b_{l,t+1}) < F(b_{l,t})$;

Signalons que tous les codes utilisés dans cette section peuvent être obtenus par simple demande à l'adresse mail *romain.hendrickx@gmail.com*.

4.4 Résultats

4.4.1 Exécutions basiques

Avant de présenter les résultats de l'exécution de l'algorithme, donnons une stratégie aléatoire, et regardons son comportement lors d'une session de nettoyage. Pour avoir une représentation raisonnable, nous donnons l'individu sous forme codée, en notant chaque action l'une à la suite de l'autre :
$b = [52115412416623644335664526717116623653225657363436374775357255$
$217414156712126251316653557766357225675262626347213556331634714436$
$554711454273367537161354435624727 2]$

Si l'on simule une session de nettoyage avec cette stratégie, on se rend compte que celle-ci n'est pas du tout performante. En effet, selon la disposition des canettes, plusieurs comportements "stupides" sont constatés, par exemple le fait de foncer continuellement dans le mur situé au nord, ou de rester à la même place pendant les 200 actions, ou encore de descendre d'une case au sud et puis de foncer dans le mur en ouest. La récompense moyenne ramassée sur 50 sessions est de $f(b) = 0.40$. Comme la fitness est normalisée dans [0, 1], cela peut être interprété comme une solution dont la qualité est de 40%.

Cependant, pour avoir une idée du comportement de la stratégie, on peut mettre en avant certains seuils de qualité qui classent les différents types de stratégies. Par exemple, il est intéressant de savoir quand une stratégie n'a reçu aucune pénalité. Il suffit pour cela de poser $\tilde{f} \geq 0$ dans l'équation (4.3) pour trouver le seuil de $f \geq \frac{2}{3}$. Ensuite, comme chaque canette ramassée augmente la récompense $\tilde{f}$ de 10 unités, on peut calculer que la fonction normalisée augmentera respectivement de $\frac{1}{150} \simeq 0.00667$. Cela permet d'avoir une idée du nombre de canettes ramassées à partir de la fitness. Le tableau 4.4 reprend par exemple les seuils pour les tranches de 10 canettes. On voit donc que la stratégie aléatoire est en dessous du seuil de non pénalité, et constitue donc une mauvaise stratégie.

Nombre de canettes	Fitness f
0 canette	0.667
10 canettes	0.734
20 canettes	0.8
30 canettes	0.867
40 canettes	0.934
50 canettes	1

TABLE 4.4 – Tableau des correspondances entre le nombre de canettes ramassées et la fitness normalisée correspondante.

Présentons maintenant les résultats de l'exécution de l'algorithme 4.1. Pour savoir quels paramètres utiliser, nous nous basons dans un premier temps sur les ordres de grandeurs donnés par [Leclercq, 2010] pour chaque paramètre :

- pour la taille de la population : entre $m = 20$ et $m = 100$;
- pour la probabilité de reproduction : entre $p_C = 0.5$ et $p_C = 0.7$;
- pour la probabilité de mutation : entre $p_M = 0.001$ et $p_M = 0.01$.

Nous avons choisi en particulier les valeurs $m = 50$, $p_C = 0.7$ et $p_M = 0.01$. Nous avons également choisi des versions basiques pour les opérateurs de sélection, reproduction et mutation, à savoir respectivement la roulette, le 1-crossover et la 1-inversion.

La figure 4.2 représente l'évolution de la fitness au cours des générations, pour le meilleur individu (en bleu) et pour la moyenne sur toute la population (en rouge). On voit sur cette figure que l'algorithme converge très vite vers la valeur $f = 0.67047$ (après 105 générations, ou 82.4 secondes en temps CPU), mais qu'ensuite celui-ci n'évolue plus et que l'algorithme stagne dans le maximum local trouvé. En comparant la valeur maximale trouvée avec les seuils du tableau 4.4, on peut voir que l'algorithme a permis d'éviter les comportements "stupides" dans les stratégies, mais ne permet pas de ramasser beaucoup de canettes : en moyenne 1 canette pour la meilleure stratégie trouvée.

Le problème rencontré ici est ce qu'on appelle une *convergence prématurée*, et est due au mauvais ajustement des paramètres. En effet, la sélection est trop forte et augmente trop vite l'homogénéité de la population, empêchant une exploration efficace de l'espace des solutions par les opérateurs de reproduction et de mutation. Pour renforcer cette affirmation, nous avons tracé sur la figure 4.3 l'évolution de la *dispertion* des individus de la population, qui est calculée comme la moyenne de la distance de Hamming entre les individus de la population pris deux à deux. La distance de Hamming est simplement la fonction qui calcule le nombre de composantes différentes entre deux vecteurs. Notons que nous avons divisé le résultat obtenu par le nombre de gènes n, de manière à obtenir une valeur dans l'intervalle [0, 1]. On peut voir sur cette figure que la dispersion de la population décroit très rapidement, et devient inférieure à 10^{-2} à partir de l'itération 50. La valeur à la dernière génération est de 0.00317. Cela signifie qu'en moyenne, lorsqu'on prend deux individus de la population au hasard, ils ont environ $0.00317n \simeq 0.5$ gènes de différent. La population est donc composée de m version d'un même individu, avec parfois 1 gène différent. Il est donc clair que les paramètres choisis pour le GA ne sont pas adéquats.

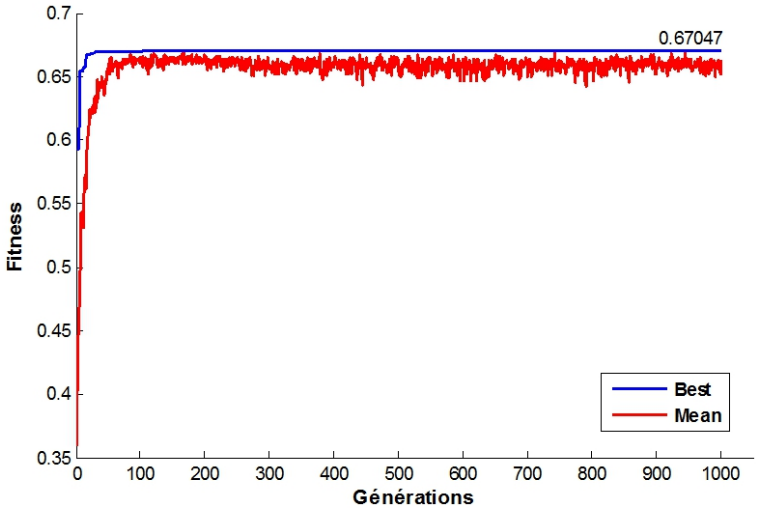


FIGURE 4.2 – Évolution de la fitness au cours des générations lors de l'exécution du GA pour $m = 50$, S=1, R=1, M=1, $p_C = 0.7$ et $p_M = 0.01$.

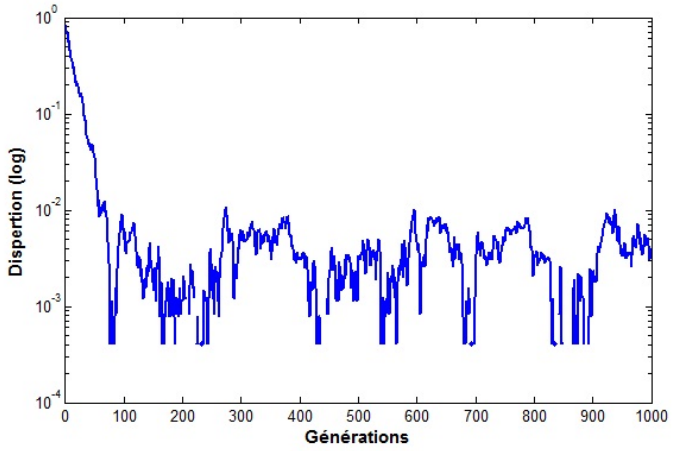


FIGURE 4.3 – Évolution de la dispersion de la population au cours des générations lors de l'exécution du GA pour $m = 50$, S=1, R=1, M=1, $p_C = 0.7$ et $p_M = 0.01$.

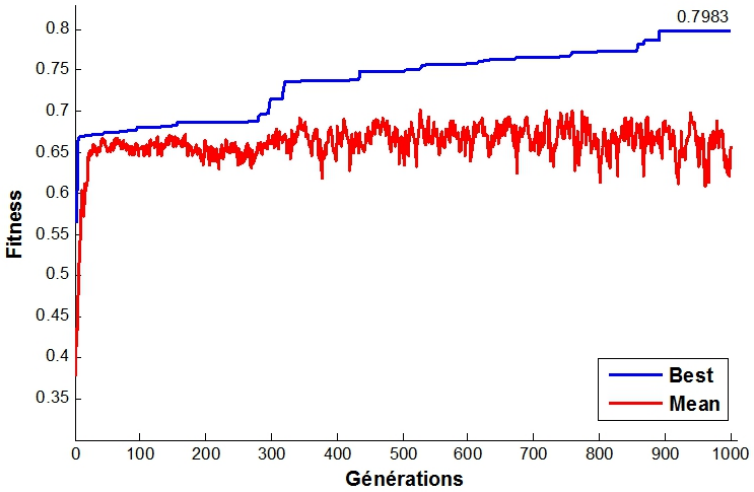


FIGURE 4.4 – Évolution de la fitness au cours des générations lors de l'exécution du GA pour $m = 50$, S=3, R=3, M=2, $p_C = 0.7$, $p_T = 0.7$ et $p_M = 0.01$.

Pour éviter ce problème et pour trouver de meilleures solutions, il faut donc trouver le bon jeu de paramètres et d'opérateurs. Cependant, cette tâche constitue un problème d'optimisation en lui-même : l'optimisation des performances de l'algorithme par rapport à ses paramètres. Ce n'est toutefois pas notre but premier, et il serait de toute façon assez irréaliste d'appliquer un algorithme d'optimisation sur lui-même pour ce problème, vu le nombre de paramètres et le temps que prendrait chaque évaluation de la fonction objectif du méta-algorithme, puisqu'elle correspondrait à une exécution complète du GA sur le problème initial.

Une autre solution est de tenter la stratégie par "essais-erreurs", qui consiste à essayer manuellement plusieurs jeux de paramètres et voir les performances du GA associées. Par exemple, en choisissant les opérateurs de crossover uniforme, de sélection par tournoi, de mutation par inversion aléatoire, et des paramètres $m = 50$, $p_C = 0.7$, $p_M = 0.01$ et $p_T = 0.7$, on obtient le résultat donné par la figure 4.4. On peut voir qu'il y a une nette amélioration par rapport à l'exécution précédente puisque la solution trouvée possède une fitness de $f = 0.7983$ après $t_{\max} = 1000$ itérations, ce qui correspond au ramassage d'environ 19.7 canettes sur les 50.

Cependant, on espère que la solution peut encore être améliorée, et qu'il est possible de trouver une stratégie qui permette de ramasser plus de $\sim$40% des canettes. Il faut donc laisser tourner le GA "assez longtemps" afin qu'il trouve une solution qui s'approche le plus possible de l'optimum global. Ce-

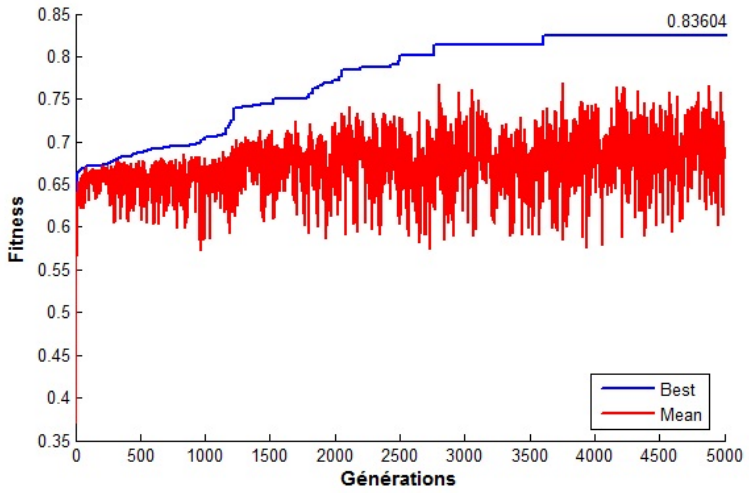


FIGURE 4.5 – Évolution de la fitness au cours des générations lors de l'exécution du GA pour $m = 50$, S=3, R=3, M=2, $p_C = 0.7$, $p_T = 0.7$ et $p_M = 0.01$, pour une longue exécution (5000 itérations, environ 5 heures).

pendant, comme le montre la figure 4.5, laisser tourner le GA longtemps n'est pas suffisant pour trouver une solution proche de l'optimum. Nous avons en effet laissé le GA tourner pendant un peu plus de 5 heures en gardant les mêmes paramètres qu'avant. La solution trouvée à l'issue d'une telle longue exécution possède une fonction objectif de 0.83604, soit environ 50% de canettes ramassées, ce qui n'est pas encore très convaincant. De plus, on peut voir sur cette figure que l'algorithme stagne fortement dans les dernières itérations, et une forte augmentation semble peu probable, et ce même si on laisse tourner l'algorithme plus longtemps. Il est donc nécessaire de dégager un bon jeu de paramètres de manière à diminuer autant que possible le temps global pris par l'algorithme et de trouver une solution d'assez bonne qualité.

4.4.2 Étude des paramètres

Cette section a pour but de tenter de mettre en avant un meilleur jeu de paramètres pour l'algorithme 4.1. Comme on ne peut (veut) pas effectuer une *optimisation* des paramètres en tant que telle, il faut trouver une méthode peu coûteuse qui permette de révéler les meilleurs jeux de paramètres. Cette section permet également de donner un aperçu de l'influence des différents paramètres et opérateurs sur les performances du GA pour ce problème.

La méthode que nous avons utilisée consiste à comparer les fitness des solutions trouvées par le GA pour plusieurs jeux de paramètres donnés, en fixant un certain temps d'exécution pour le GA, et de choisir ensuite les paramètres qui amènent à la meilleure fitness pour le temps fixé. Le but est

Paramètre	Valeurs considérées
Taille de la population	$m = 30$ ou $m = 50$
Stratégie de sélection	S=1 ou S=2 ou S=3
Stratégie de reproduction	R=1 ou R=2 ou R=3
Stratégie de mutation	M=1 ou M=2
Probabilité de reproduction	$p_C = 0.5$ ou $p_C = 0.7$ ou $p_C = 0.9$
Probabilité de mutation	$p_M = 0.001$ ou $p_M = 0.01$ ou $p_M = 0.05$
Probabilité de tournoi (quand S=3)	$p_T = 0.6$ ou $p_T = 0.7$ ou $p_T = 0.8$

TABLE 4.5 – Valeurs considérées pour les paramètres et opérateurs.

de trouver un GA qui soit le plus *efficace* possible, c'est-à-dire qui donne les meilleurs résultats en un temps fixé, l'espoir étant que celui-ci reste efficace après le temps limite et amène le plus vite possible à une solution d'assez bonne qualité. Évidemment, il faut choisir le temps limite pour l'exécution du GA assez grand, car sinon le risque est de choisir des paramètres qui amènent à une convergence prématurée, et donc une stagnation de l'algorithme par la suite. Par contre, il ne doit pas non plus être trop grand, de manière à éviter que la recherche du meilleur jeu de paramètres ne soit trop longue. Comme il apparaît clairement sur la figure 4.2 que 300 exécutions sont suffisantes pour déterminer si l'algorithme converge prématurément, on choisit de fixer le temps d'exécution du GA à 15 minutes, puisque c'est approximativement le temps pris par l'algorithme pour effectuer ce nombre d'itérations. Notons qu'il est important de considérer le *temps d'exécution* et non le *nombre de générations* car ceux-ci seront potentiellement différents pour des choix d'opérateurs ou de paramètres différents, en particulier pour des tailles de populations m différentes.

Pour limiter le temps de recherche total, nous ne considérons que quelques valeurs pour chacun des paramètres. Celles-ci ne sont pas déterminées totalement arbitrairement, mais sont basées plus ou moins sur les recommandations données par [Leclercq, 2010]. Les valeurs considérées pour chacun des paramètres sont reprises dans le tableau 4.5.

Le nombre de combinaisons possibles pour l'ensemble de ces valeurs est de 540 possibilités. Comme nous avons fixé un temps d'exécution du GA de 15 minutes, cela signifie qu'il faudrait $540 \cdot 15$ minutes $= 8100$ minutes, soit plus de 5 jours, pour obtenir les résultats des GAs pour l'ensemble de ces combinaisons de paramètres. Nous décidons donc de ne pas considérer toutes les combinaisons possibles, mais plutôt de séparer l'analyse en comparant d'une part les performances des opérateurs et de la taille de la population, et d'autre part les performances des paramètres p_C, p_M et p_T. Cela permet de réduire l'analyse à quelques heures au lieu de quelques jours.

Performances des opérateurs et de la taille de la population

Considérons donc dans un premier temps les opérateurs de sélection, de reproduction et de mutation, ainsi que la taille de la population, en fixant les autres paramètres. Notons que nous choisissons les mêmes valeurs de paramètres qu'auparavant, *i.e.* $p_C = 0.7$, $p_M = 0.01$ et $p_T = 0.7$. Le tableau 4.6 reprend les résultats des exécutions du GA pour les différentes combinaisons d'opérateurs/taille de population considérées. Nous avons présenté ceux-ci dans un tableau différent pour chaque critère, de manière à rendre l'analyse plus facile.

Considérons tout d'abord le paramètre de la taille de population. Nous avons considéré deux valeurs : $m = 30$ et $m = 50$. Les résultats sont repris dans le tableau 4.6(1). Nous avons présenté les résultats en deux colonnes, la première pour $m = 30$ et la deuxième pour $m = 50$, où chaque ligne correspond à une combinaison possible des opérateurs. Pour chaque ligne, nous avons écrit en vert le meilleur résultat entre l'exécution avec $m = 30$ et celle avec $m = 50$. Sur l'ensemble des exécutions effectuées, la valeur $m = 30$ a une moyenne des fitness obtenues *légèrement* plus élevée, et est meilleure que la valeur $m = 50$ dans 10 cas sur 18. Cependant, les deux meilleures exécutions sont rencontrées avec une valeur de $m = 50$ (8^e et 12^e lignes). On ne conclut donc pas à une différence significative entre les deux valeurs de paramètres.

Comparons maintenant les performances des opérateurs de mutation. Les résultats pour les deux stratégies de mutation sont donnés dans le tableau 4.6(2). Nous avons de nouveau écrit les meilleurs résultats en vert. Il est clair cette fois qu'une stratégie est meilleure que l'autre. En effet, la stratégie de mutation par *inversion aléatoire* (M=2) est meilleure que la stratégie par 1−*inversion* (M=1), et ce pour *toutes* les combinaisons des autres paramètres. Nous pouvons donc conclure que dans le cadre du problème considéré, et pour les valeurs des paramètres fixés, la mutation par inversion aléatoire est meilleure que la 1−inversion. Il faut toutefois émettre des réserves par rapport à cette conclusion. En effet, ce qui influe réellement sur les performances de l'algorithme pour la mutation est le taux de gènes changés à chaque génération. Ce taux varie évidemment avec la probabilité d'application p_M, mais aussi avec l'opérateur de mutation considéré. Par exemple, l'espérance du nombre de gènes modifiés pour l'opérateur de 1−inversion vaut p_M, puisque chaque *individu* a une probabilité p_M d'avoir l'*un* de ses gènes modifié, alors qu'il vaut $np_M = 162p_M$ pour l'opérateur d'inversion aléatoire, puisque dans ce cas c'est chaque *gène* qui a une probabilité p_M d'être modifié. Pour que les deux opérateurs soient comparables il faudrait donc considérer des valeurs de p_M qui soient 162 fois plus grandes pour la 1−inversion que pour l'inversion aléatoire. Cependant, à partir d'une valeur

de $p_M = \frac{1}{162}$ pour l'inversion aléatoire, la probabilité correspondante pour la mutation par 1−inversion atteint $p_M = 1$ et tous les individus sont mutés d'un gène, ce qui est potentiellement faible lorsque la taille de la population est petite. Nous privilégions donc l'utilisation de la mutation par inversion aléatoire, qui est moins limitée que la 1−inversion.

En ce qui concerne l'opérateur de sélection, on peut voir dans le tableau 4.6(3) que globalement la stratégie de sélection par rang est meilleure que les deux autres stratégies. En effet, celle-ci est meilleure 11 fois sur les 12 combinaisons des autres opérateurs et de la taille de la population. Cette fois, il ne peut y avoir de doute que par rapport la sélection par tournoi, car les deux autres stratégies ne dépendent pas de paramètres. Mais pour la valeur choisie $p_T = 0.7$, la sélection par rang amene à des résultats sensiblement meilleurs que les deux autres. Notons que ces derniers ont quant à eux des performances semblables.

Enfin, pour l'opérateur de reproduction, nous ne pouvons pas vraiment tirer de conclusion sur le fait d'avoir un meilleur opérateur. Comme pour les autres comparaisons, le tableau des résultats est donné sur la figure 4.6(4). Il semblerait que les opérateurs de 1−crossover et de crossover uniforme aient des performances à peu près identiques, et que le 2−crossover soit légèrement moins bon. En effet, lorsqu'on regarde le nombre de fois où chaque stratégie de sélection est la meilleure sur les trois stratégies pour la combinaison particulière des autres paramètres, on obtient que le 1−crossover et le crossover uniforme sont tous deux 5 fois les meilleures stratégies, contre seulement 2 fois pour le 2−crossover. Cependant, lorsqu'on regarde les moyennes des fitness obtenues, celles-ci sont presque identiques. Nous ne tirons donc pas de conclusion sur le fait qu'un opérateur soit sensiblement meilleur que les autres.

Remarquons que le meilleur résultat obtenu sur l'ensemble des combinaisons considérées a été obtenu avec les paramètres S=2, R=1, M=2 et $m = 50$. La fitness après 15 minutes d'exécution est de 0.86677, ce qui constitue environ 60% des canettes ramassées. Ce résultat est déjà bien meilleur que les deux résultats de la section précédente. Il semblerait donc que ce jeu de paramètres constitue une bonne base pour le jeu de paramètres qui sera utilisé pour une longue exécution du GA. Avant cela, analysons l'influence des différents paramètres restants.

Performance des probabilités p_C, p_M et p_T

Étudions maintenant les performances liées aux probabilités pour chaque opérateur. Pour ce faire, reprenons la meilleure combinaison d'opérateurs et taille de population mise en évidence dans la section précédente, *i.e.* une

S	R	M	$m = 30$	$m = 50$
1	1	1	0.67160	0.66743
1	1	2	0.69906	0.67880
1	2	1	0.67047	0.66987
1	2	2	0.71427	0.68333
1	3	1	0.67140	0.66993
1	3	2	0.70527	0.70107
2	1	1	0.66947	0.67213
2	1	2	0.84113	0.86677
2	2	1	0.67113	0.68120
2	2	2	0.83193	0.79723
2	3	1	0.67473	0.67500
2	3	2	0.79646	0.85040
3	1	1	0.66881	0.66287
3	1	2	0.75017	0.71370
3	2	1	0.66760	0.67060
3	2	2	0.70060	0.69780
3	3	1	0.67113	0.67400
3	3	2	0.67960	0.70700
Moyenne			0.70860	0.70770
Nombre >			10	8

(1)

S	R	m	M=1	M=2
1	1	30	0.67160	0.69906
1	1	50	0.66743	0.67880
1	2	30	0.67047	0.71427
1	2	50	0.66987	0.68333
1	3	30	0.67140	0.70527
1	3	50	0.66993	0.70107
2	1	30	0.66947	0.84113
2	1	50	0.67213	0.86677
2	2	30	0.67113	0.83193
2	2	50	0.68120	0.79723
2	3	30	0.67473	0.79646
2	3	50	0.67500	0.85040
3	1	30	0.66881	0.75017
3	1	50	0.66287	0.71370
3	2	30	0.66760	0.70060
3	2	50	0.67060	0.69780
3	3	30	0.67113	0.67960
3	3	50	0.67400	0.70700
Moyenne			0.6711	0.7453
Nombre >			0	18

(2)

R	M	m	S=1	S=2	S=3
1	1	30	0.67160	0.66947	0.66881
1	1	50	0.66743	0.67213	0.66287
1	2	30	0.69906	0.84113	0.75017
1	2	50	0.67880	0.86677	0.71370
2	1	30	0.67047	0.67113	0.66760
2	1	50	0.66987	0.68120	0.67060
2	2	30	0.71427	0.83193	0.70060
2	2	50	0.68333	0.79723	0.69780
3	1	30	0.67140	0.67473	0.67113
3	1	50	0.66993	0.67500	0.67400
3	2	30	0.70527	0.79646	0.67960
3	2	50	0.70107	0.85040	0.70700
Moyenne			0.6835	0.7523	0.6886
Nombre >			1	11	0

(3)

R	M	m	R=1	R=2	R=3
1	1	30	0.67160	0.67047	0.67140
1	1	50	0.66743	0.66987	0.66993
1	2	30	0.69906	0.71427	0.70527
1	2	50	0.67880	0.68333	0.70107
2	1	30	0.66947	0.67113	0.67473
2	1	50	0.67213	0.68120	0.67500
2	2	30	0.84113	0.83193	0.79646
2	2	50	0.86677	0.79723	0.85040
3	1	30	0.66881	0.66760	0.67113
3	1	50	0.66287	0.67060	0.67400
3	2	30	0.75017	0.70060	0.67960
3	2	50	0.71370	0.69780	0.70700
Moyenne			0.7135	0.7047	0.7063
Nombre >			5	2	5

(4)

TABLE 4.6 – Résultats de l'exécution du GA avec un temps fixé de 15 minutes (temps CPU). Le tableau (1) présente les résultats pour la taille de population m, le tableau (2) pour l'opérateur de mutation, le tableau (3) pour l'opérateur de sélection et le tableau (4) pour l'opérateur de reproduction.

sélection par rang (S=2), une reproduction par 1-crossover (R=1) et une mutation par inversion aléatoire (M=2). On décide de considérer également l'opérateur de sélection par tournoi pour étudier le paramètre p_T. Nous procédons de manière analogue à la section précédente, c'est-à-dire nous présentons dans des tableaux les résultats obtenus pour chaque paramètre étudié. Ces tableaux sont repris sur la figure 4.7.

Examinons d'abord les performances liées aux stratégies de sélection par rang et par tournoi (pour lequel on fait varier les probabilités p_T associées). Les fitness des exécutions du GA à temps fixe (de 15 minutes temps CPU) pour les différentes valeurs des paramètres sont reprises dans le tableau 4.7(1). Ces résultats confirment le fait que la stratégie par rang est meilleure que les autres stratégies, car dans presque tous les cas (7 cas sur 9) celle-ci est plus efficace que la stratégie par tournoi (pour n'importe quelle valeur du paramètre p_T). De plus, la moyenne des fitness obtenues lors de chaque exécution du GA est sensiblement plus élevée pour la stratégie par rang que pour la stratégie par tournoi. Notons que pour cette dernière stratégie, la moyenne ne change pas beaucoup (ou pas autant) suivant les valeurs considérées de la probabilité p_T. Il semblerait toutefois que la probabilité de $p_T = 0.7$ amène à des résultats légèrement meilleurs que les autres valeurs.

Considérons ensuite la probabilité de reproduction p_C. Les résultats des exécutions du GA sont repris dans le tableau 4.7(2). Cette fois la différence de performance entre les différentes valeurs n'est pas catégorique. Seule la valeur de $p_C = 0.9$ semble amener à des résultats un peu moins bons. Pour les valeurs $p_C = 0.5$ et $p_C = 0.7$, les moyennes des fitness obtenues sont presque identiques, avec un petit avantage pour la dernière valeur. En effet, la valeur de $p_C = 0.7$ donne les meilleurs résultats dans 6 cas sur 12, contre 4 cas sur 12 pour $p_C = 0.5$ (et 2 cas sur 12 pour $p_C = 0.9$).

Analysons enfin les résultats obtenus pour la probabilité de mutation p_M. Ceux-ci sont donnés dans le tableau 4.7(3). La valeur qui donne les meilleures performances dans presque tous les cas (10 cas sur 12) est la probabilité $p_M = 0.01$. Remarquons que les moyennes des fitness obtenues sont assez différentes d'une valeur à l'autre du paramètre. La moyenne la plus haute est évidemment obtenue pour la valeur $p_M = 0.01$, la deuxième pour $p_M = 0.05$ et la dernière pour $p_M = 0.001$. La valeur de la probabilité de mutation est donc un paramètre très important qui peut changer complètement les performances de l'algorithme. Lorsqu'elle est trop faible, l'algorithme se retrouve vite bloqué et présente de longues périodes de stagnation. Par contre, une valeur trop élevée rend la recherche trop aléatoire. Il faut donc une balance entre ces deux extrêmes, ce qui semble être réalisé par la valeur $p_M = 0.01$ dans notre cas. Notons que la valeur de p_M qui donne les meilleurs résultats est la borne supérieur conseillée par [Leclercq, 2010],

p_C	p_M	S=2	S=3, $p_T = 0.6$	S=3, $p_T = 0.7$	S=3, $p_T = 0.8$
0.5	0.001	0.72571	0.69087	0.67813	0.70033
0.5	0.01	0.78073	0.70000	0.76040	0.67867
0.5	0.05	0.68327	0.67767	0.68167	0.67467
0.7	0.001	0.67133	0.69580	0.70147	0.68715
0.7	0.01	0.85844	0.71573	0.70960	0.70973
0.7	0.05	0.70047	0.67267	0.67413	0.68153
0.9	0.001	0.69793	0.67927	0.68067	0.67087
0.9	0.01	0.70353	0.68040	0.72780	0.72460
0.9	0.05	0.68554	0.68247	0.67967	0.67333
Moyenne		0.72299	0.68832	0.69928	0.68898
Nombre >		7	0	2	0

(1)

p_M	S/p_T	$p_C = 0.5$	$p_C = 0.7$	$p_C = 0.9$
0.001	2/-	0.72571	0.67133	0.69793
0.001	3/0.6	0.69087	0.6958	0.67927
0.001	3/0.7	0.67813	0.70147	0.68067
0.001	3/0.8	0.70033	0.68715	0.67087
0.01	2/-	0.78073	0.85844	0.70353
0.01	3/0.6	0.70000	0.71573	0.68040
0.01	3/0.7	0.76040	0.70960	0.72780
0.01	3/0.8	0.67867	0.70973	0.72460
0.05	2/-	0.68327	0.70047	0.68554
0.05	3/0.6	0.67767	0.67267	0.68247
0.05	3/0.7	0.68167	0.67413	0.67967
0.05	3/0.8	0.67467	0.68153	0.67333
Moyenne		0.70268	0.70651	0.69050
Nombre >		4	6	2

(2)

p_C	S/p_T	$p_M = 0.001$	$p_M = 0.01$	$p_M = 0.05$
0.5	2/-	0.72571	0.78073	0.68327
0.5	3/0.6	0.69087	0.70000	0.67767
0.5	3/0.7	0.67813	0.76040	0.68167
0.5	3/0.8	0.70033	0.67867	0.67467
0.7	2/-	0.67133	0.85844	0.70047
0.7	3/0.6	0.69580	0.71573	0.67267
0.7	3/0.7	0.70147	0.70960	0.67413
0.7	3/0.8	0.68715	0.70973	0.68153
0.9	2/-	0.69793	0.70353	0.68554
0.9	3/0.6	0.67927	0.68040	0.68247
0.9	3/0.7	0.68067	0.72780	0.67967
0.9	3/0.8	0.67087	0.72460	0.67333
Moyenne		0.63323	0.72914	0.68059
Nombre >		1	10	1

(3)

TABLE 4.7 – Résultats de l'exécution du GA avec un temps fixé de 15 minutes (temps CPU). Le tableau (1) présente les résultats pour la probabilité de tournoi p_T, le tableau (2) pour la probabilité de crossover p_C et le tableau (3) pour la probabilité de mutation p_M.

et qu'une probabilité $p_M = 0.05$, qui est encore plus élevée, semble donner de meilleurs résultats que la probabilité $p_M = 0.01$. Cela peut s'expliquer par le fait que nous considérons dans notre représentation un ensemble de gènes qui représente des états impossibles, et qui n'ont donc pas d'impact sur la stratégie associée. En effet, cela a pour conséquence de réduire le taux de mutation sur les gènes *effectifs*, et dès lors il semble clair que le seuil de $p_M = 0.001$ devient vraiment trop faible.

Pour conclure cette section, remarquons que suivant les opérateurs, c'est le *type* d'opérateur et/ou la valeur du *paramètre* de cet opérateur qui influence(nt) l'efficacité du GA. Pour la mutation, il semble clair que c'est surtout le taux de mutation général qui importe, qui est contrôlé comme

nous l'avons déjà fait remarquer par le type d'opérateur *et* la probabilité associée. Il faut donc trouver le bon équilibre compte tenu de ces deux critères. Par contre, il semble que les performances du GA ne sont pas fortement influencées par le choix particulier d'opérateur de reproduction. Il faut juste veiller à ce que la probabilité de reproduction soit dans des bornes raisonnables. Enfin, l'opérateur de sélection est sans doute le plus délicat à choisir, et il semble important d'éviter la stratégie basique de la roulette. Ceci dit, peut être que le couplage avec une stratégie de mutation ou de reproduction adaptée rendrait son utilisation réaliste. En effet, l'efficacité du GA dépend de l'*ensemble* des paramètres de manière *croisée*, et certains opérateurs se complètent alors que d'autres sont plus incompatibles.

Soulignons le fait que la vision que nous avons donnée ici est donc incomplète, et qu'il serait intéressant d'étudier plus en profondeur le rôle et les relations qu'entretiennent les opérateurs et les paramètres entre-eux. Cependant, nous sommes arrivés à notre but initial, qui consistait à dégager un meilleur jeu de paramètres pour le GA. Nous pouvons maintenant envisager une longue exécution de l'algorithme pour trouver la meilleure solution possible.

4.4.3 Résultats et interprétation

Nous pouvons donc envisager une longue exécution du GA sur le problème de stratégie de nettoyage du robot. Nous utilisons pour cela la version du GA qui amène aux meilleures performances sur l'ensemble des combinaisons considérées dans la section précédente. Pour rappel, la meilleure combinaison que nous avons trouvée est formée par la stratégie de sélection par rang, la reproduction par $1-$crossover, la mutation par inversion aléatoire, et par les valeurs $m = 50$, $p_C = 0.7$ et $p_M = 0.01$ pour les autres paramètres.

La figure 4.6 représente l'exécution avec l'algorithme ainsi paramétré. Nous avons laissé tourner celui-ci pendant 5000 générations, soit environ 5 heures (18473 secondes) en temps CPU, avec un processeur Intel Core i3 2.13 GHz. La valeur atteinte pour la fonction objectif est de 0.97033, ce qui représente une moyenne d'environ 91% de canettes ramassées. Remarquons déjà le net changement de performance du GA par rapport au jeu de paramètres considéré dans la section 4.4.1, où l'on avait obtenu pour un même nombre de générations et un temps similaire une fitness maximale de 0.8360.

En observant la figure 4.6, on peut voir que la convergence du GA est caractérisée par deux phases. Dans un premier temps, l'algorithme croît la fitness du meilleur individu de manière assez régulière, de telle manière à trouver une très bonne solution après 1180 générations (environ 1h10). La fitness du meilleur individu à ce moment est en effet de 0.9597, ce qui cor-

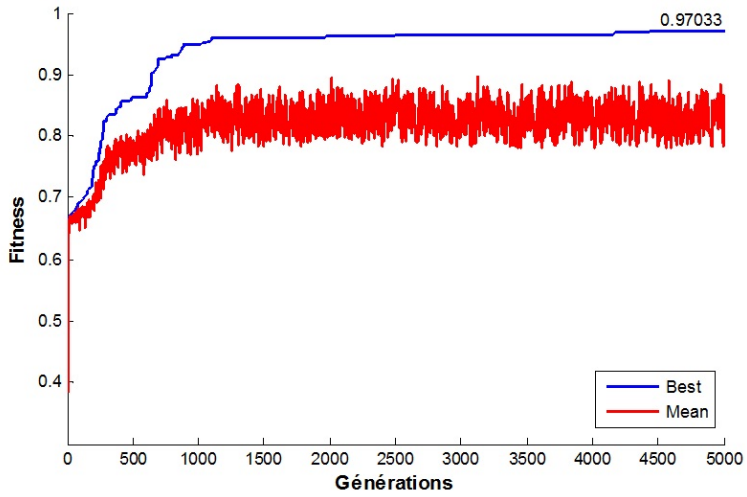


FIGURE 4.6 – Évolution de la fitness au cours des générations lors de l'exécution du GA pour $m = 50$, S=2, R=1, M=2, $p_C = 0.7$ et $p_M = 0.01$.

respond à une moyenne d'environ 88% de canettes ramassées. Ensuite la convergence devient *très* lente, et est caractérisée par de faibles améliorations *ponctuelles* au cours des générations. On peut calculer que le taux de croissance dans la seconde partie est un peu plus de 100 fois plus faible que dans la première. Cela traduit le fait qu'à partir d'un certain moment, comme la solution est déjà très bonne, les améliorations trouvées sont de plus en plus coûteuses pour l'algorithme.

Tentons maintenant d'analyser la solution obtenue à l'issue de l'exécution de l'algorithme. Une représentation de cette stratégie est donnée sur la figure 4.7. Nous avons listé sur cette figure l'ensemble des états possibles dans lesquels le robot peut se retrouver, en symbolisant les murs par "|", la présence d'une canette par "C" et une case vide par "∅", et associant à chacun de ces états l'action effectuée par le robot, en symbolisant "^^^^" un mouvement au nord, "vvvv" un mouvement au sud, "<—" un mouvement à l'ouest, "—>" un mouvement à l'est, " ? ? ? ?" un mouvement au hasard, ".." une inaction et enfin "TAKE" la prise de canette.

Regardons d'abord si l'algorithme évite les comportements stupides tels que foncer dans un mur, rester sur place ou tenter de ramasser des canettes là où il n'y a rien. Pour vérifier cela, filtrons d'abord tous les états qui contiennent au moins un mur, mais qui constituent des états réalisables. Le résultat de ce filtre est repris sur la figure 4.8. Nous avons séparé les états possibles contenant les différents murs dans des tableaux différents.

Curr.	North	South	West	East	Action
Ø	Ø	Ø	Ø	Ø	????
Ø	Ø	Ø	Ø	C	--->
Ø	Ø	Ø	Ø	|	^^^^
Ø	Ø	Ø	C	Ø	<---
Ø	Ø	Ø	C	C	--->
Ø	Ø	Ø	C	|	<---
Ø	Ø	Ø	|	Ø	vvvv
Ø	Ø	Ø	|	C	--->
Ø	Ø	Ø	|	|	TAKE
Ø	Ø	C	Ø	Ø	????
Ø	Ø	C	Ø	C	vvvv
Ø	Ø	C	Ø	|	????
Ø	Ø	C	C	Ø	<---
Ø	Ø	C	C	C	^^^^
Ø	Ø	C	C	|	????
Ø	Ø	C	|	Ø	vvvv
Ø	Ø	C	|	C	vvvv
Ø	Ø	C	|	|	--->
Ø	Ø	|	Ø	Ø	--->
Ø	Ø	|	Ø	C	--->
Ø	Ø	|	Ø	|	^^^^
Ø	Ø	|	C	Ø	????
Ø	Ø	|	C	C	--->
Ø	Ø	|	C	|	<---
Ø	Ø	|	|	Ø	????
Ø	Ø	|	|	C	--->
Ø	Ø	|	|	|	--->
Ø	C	Ø	Ø	Ø	^^^^
Ø	C	Ø	Ø	C	^^^^
Ø	C	Ø	Ø	|	????
Ø	C	Ø	C	Ø	<---
Ø	C	Ø	C	C	????
Ø	C	Ø	C	|	<---
Ø	C	Ø	|	Ø	--->
Ø	C	Ø	|	C	--->
Ø	C	Ø	|	|	--->
Ø	C	C	Ø	Ø	????
Ø	C	C	Ø	C	^^^^
Ø	C	C	Ø	|	????
Ø	C	C	C	Ø	--->
Ø	C	C	C	C	????
Ø	C	C	C	|	^^^^
Ø	C	C	|	Ø	????
Ø	C	C	|	C	????
Ø	C	C	|	|	^^^^
Ø	C	|	Ø	Ø	^^^^
Ø	C	|	Ø	C	^^^^
Ø	C	|	Ø	|	^^^^
Ø	C	|	C	Ø	--->
Ø	C	|	C	C	<---
Ø	C	|	C	|	????
Ø	C	|	|	Ø	--->
Ø	C	|	|	C	????
Ø	C	|	|	|	--->

Curr.	North	South	West	East	Action
Ø	|	Ø	Ø	Ø	<---
Ø	|	Ø	Ø	C	vvvv
Ø	|	Ø	Ø	|	????
Ø	|	Ø	C	Ø	????
Ø	|	Ø	C	C	vvvv
Ø	|	Ø	C	|	<---
Ø	|	Ø	|	Ø	vvvv
Ø	|	Ø	|	C	vvvv
Ø	|	Ø	|	|	????
Ø	|	C	Ø	Ø	vvvv
Ø	|	C	Ø	C	vvvv
Ø	|	C	Ø	|	????
Ø	|	C	C	Ø	vvvv
Ø	|	C	C	C	????
Ø	|	C	C	|	vvvv
Ø	|	C	|	Ø	vvvv
Ø	|	C	|	C	vvvv
Ø	|	C	|	|	<---
Ø	|	|	Ø	Ø	..
Ø	|	|	Ø	C	TAKE
Ø	|	|	Ø	|	TAKE
Ø	|	|	C	Ø	..
Ø	|	|	C	C	vvvv
Ø	|	|	C	|	vvvv
Ø	|	|	|	Ø	vvvv
Ø	|	|	|	C	^^^^
Ø	|	|	|	|	..
C	Ø	Ø	Ø	Ø	TAKE
C	Ø	Ø	Ø	C	TAKE
C	Ø	Ø	Ø	|	TAKE
C	Ø	Ø	C	Ø	TAKE
C	Ø	Ø	C	C	<---
C	Ø	Ø	C	|	vvvv
C	Ø	Ø	|	Ø	TAKE
C	Ø	Ø	|	C	TAKE
C	Ø	Ø	|	|	^^^^
C	Ø	C	Ø	Ø	TAKE
C	Ø	C	Ø	C	TAKE
C	Ø	C	Ø	|	TAKE
C	Ø	C	C	Ø	TAKE
C	Ø	C	C	C	vvvv
C	Ø	C	C	|	^^^^
C	Ø	C	|	Ø	TAKE
C	Ø	C	|	C	TAKE
C	Ø	C	|	|	--->
C	Ø	|	Ø	Ø	TAKE
C	Ø	|	Ø	C	TAKE
C	Ø	|	Ø	|	^^^^
C	Ø	|	C	Ø	????
C	Ø	|	C	C	TAKE
C	Ø	|	C	|	TAKE
C	Ø	|	|	Ø	TAKE
C	Ø	|	|	C	TAKE
C	Ø	|	|	|	vvvv

Curr.	North	South	West	East	Action
C	C	Ø	Ø	Ø	TAKE
C	C	Ø	Ø	C	TAKE
C	C	Ø	Ø	|	TAKE
C	C	Ø	C	Ø	<---
C	C	Ø	C	C	<---
C	C	Ø	C	|	vvvv
C	C	Ø	|	Ø	TAKE
C	C	Ø	|	C	^^^^
C	C	Ø	|	|	--->
C	C	C	Ø	Ø	vvvv
C	C	C	Ø	C	vvvv
C	C	C	Ø	|	????
C	C	C	C	Ø	<---
C	C	C	C	C	<---
C	C	C	C	|	vvvv
C	C	C	|	Ø	TAKE
C	C	C	|	C	--->
C	C	C	|	|	vvvv
C	C	|	Ø	Ø	TAKE
C	C	|	Ø	C	TAKE
C	C	|	Ø	|	????
C	C	|	C	Ø	????
C	C	|	C	C	<---
C	C	|	C	|	TAKE
C	C	|	|	Ø	vvvv
C	C	|	|	C	^^^^
C	C	|	|	|	..
C	|	Ø	Ø	Ø	TAKE
C	|	Ø	Ø	C	TAKE
C	|	Ø	Ø	|	????
C	|	Ø	C	Ø	TAKE
C	|	Ø	C	C	TAKE
C	|	Ø	C	|	vvvv
C	|	Ø	|	Ø	TAKE
C	|	Ø	|	C	TAKE
C	|	Ø	|	|	--->
C	|	C	Ø	Ø	????
C	|	C	Ø	C	????
C	|	C	Ø	|	vvvv
C	|	C	C	Ø	TAKE
C	|	C	C	C	TAKE
C	|	C	C	|	--->
C	|	C	|	Ø	TAKE
C	|	C	|	C	TAKE
C	|	C	|	|	--->
C	|	|	Ø	Ø	^^^^
C	|	|	Ø	C	--->
C	|	|	Ø	|	..
C	|	|	C	Ø	vvvv
C	|	|	C	C	<---
C	|	|	C	|	^^^^
C	|	|	|	Ø	--->
C	|	|	|	C	????
C	|	|	|	|	????

FIGURE 4.7 – Représentation de la stratégie optimale trouvée par le GA.

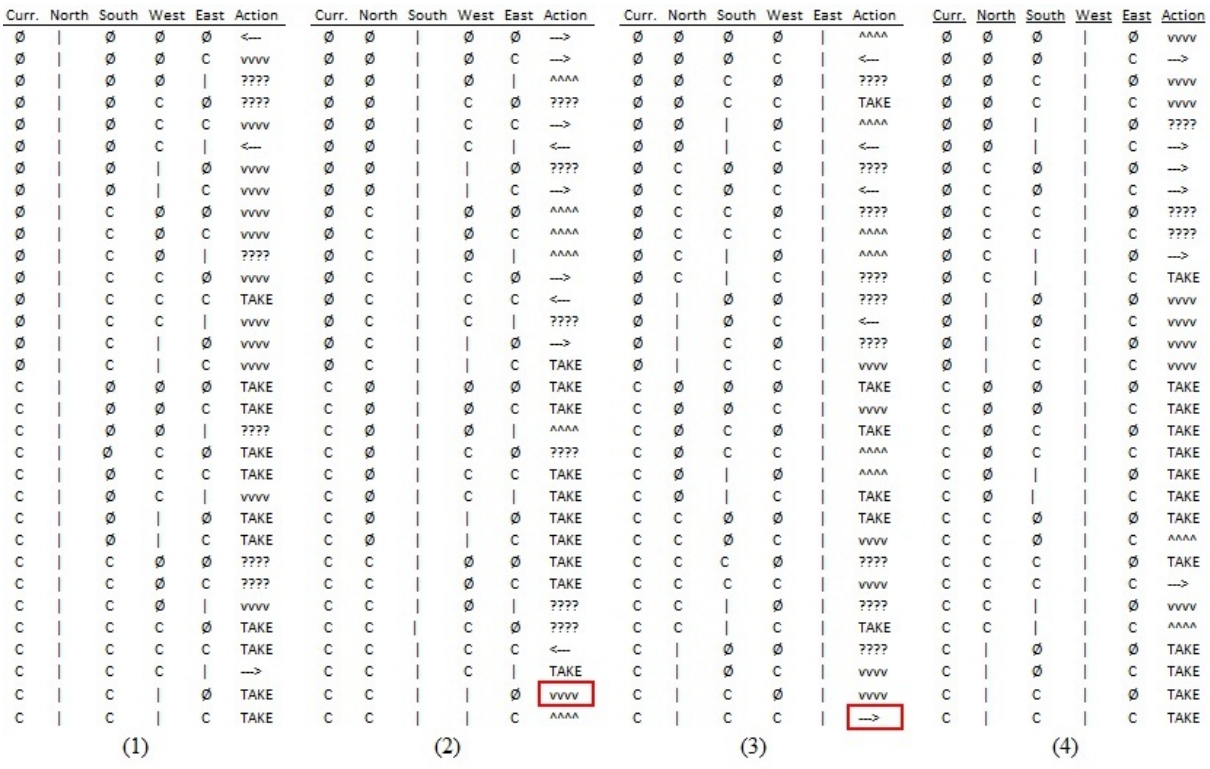


FIGURE 4.8 – Filtre des états contenant des murs pour la stratégie optimale obtenue par le GA. (1) États contenant un mur au nord, (2) un mur au sud, (3) un mur à l'est et (4) un mur à l'ouest.

Un rapide coup d'oeil permet de vérifier qu'il reste deux comportements stupides, encadrés en rouge sur la figure. Il s'agit d'états particuliers pour le coin inférieur gauche et pour le coin supérieur droit de la grille. La présence de ces comportements peut être expliquée par plusieurs raisons. Tout d'abord, il s'agit peut être d'états qui ne sont jamais atteints. En effet, les décisions prises auparavant font peut être que ces états ne sont jamais rencontrés. Pour vérifier si c'est le cas avec les états mis en avant, nous lançons 10^4 simulations de sessions de nettoyage avec des grilles différentes, et comptabilisons le nombre de fois où chaque état "problématique" est rencontré (*i.e.* le nombre de fois où le robot décide de foncer dans le mur). Sur l'ensemble des simulations, seule la deuxième situation a été rencontrée, mais seulement 14 fois. Le premier état est donc sans doute un état inaccessible de par les autres comportements, et le deuxième n'est pas rencontré très souvent, ce qui explique que la fitness moyenne ne soit pas fortement touchée par ce défaut. Une idée pour tenter de pallier à ce problème serait peut être de déterminer la position de départ du robot de manière aléatoire.

Vérifions maintenant les comportements d'inaction du robot. On peut facilement vérifier sur la figure 4.7 que les états pour lesquels une décision d'inaction est choisie sont tous impossibles, soit parce qu'ils comportent plus d'un mur, soit parce que les murs sont disposés selon deux directions cardinales opposées (un mur au nord *et* au sud par exemple). De même, on peut vérifier qu'il n'y a pas d'état possible pour lequel la case courante est vide et l'action est de ramasser la canette. La solution retournée par l'algorithme évite donc tous les comportements stupides, à une exception près, ce qui est déjà une bonne chose.

Pour faire le lien entre la croissance de la fitness et l'apprentissage du robot, nous avons calculé le nombre moyen de sessions pendant laquelle le robot se prend un mur, le nombre de sessions où le robot tente de ramasser des canettes là où il n'y a rien, le nombre de sessions où le robot reste inactif après un certain temps et enfin le nombre moyen de canettes ramassées par session, d'une part pour le meilleur individu de chaque génération et d'autre part en moyenne sur l'ensemble de la génération. Les 4 courbes résultantes sont données sur la figure 4.9. Nous pouvons remarquer que l'algorithme génétique permet d'abord d'apprendre aux individus d'éviter les murs (très nette réduction du nombre de collisions avec un murs dans les 10 premières itérations) ainsi que d'éviter de ramasser des canettes dans le vide (forte réduction dans les 40 premières itérations). Cela correspond en fait à la croissance de la fitness au seuil de 0.667 que nous avions déjà remarquée dans la section 4.4.1. En ce qui concerne le nombre de canettes ramassées, l'apprentissage semble assez progressif, tant au niveau de l'ensemble de la population que du meilleur individu. En effet, une fois que les individus ont appris à éviter de foncer dans les murs et de ramasser dans le vide, le GA permet une évolution par rapport au nombre de canettes ramassées. Et en effet, on peut vérifier que la courbe du nombre de canettes ramassées au cours des générations colle assez bien à la courbe d'évolution de la fitness 4.6, après la forte évolution des 30-40 premières générations (qu'on ne distingue pas bien sur la figure 4.6 car elle se fait sur un nombre très faible d'itérations par rapport aux 5000 itérations totales). Enfin, remarquons que la courbe de l'évolution du nombre d'inactions ne décroît pas de manière aussi brusque que le nombre de collisions ou de ramassages dans le vide. Cela peut s'expliquer par le fait qu'aucune pénalité n'est associée aux inactions. Dès lors, l'apprentissage pour éviter les inactions se fait de manière indirecte, par la préférence du GA des stratégies qui ramassent un plus grand nombre de canettes. Il serait sans doute intéressant de pénaliser a priori les inactions, pour mener plus vite à de bonnes solutions.

Tentons maintenant de comprendre la stratégie optimale trouvée qui permet au robot de ramasser plus de 90% des canettes. Remarquons tout d'abord que contrairement à ce que l'on pourrait penser, la solution ne

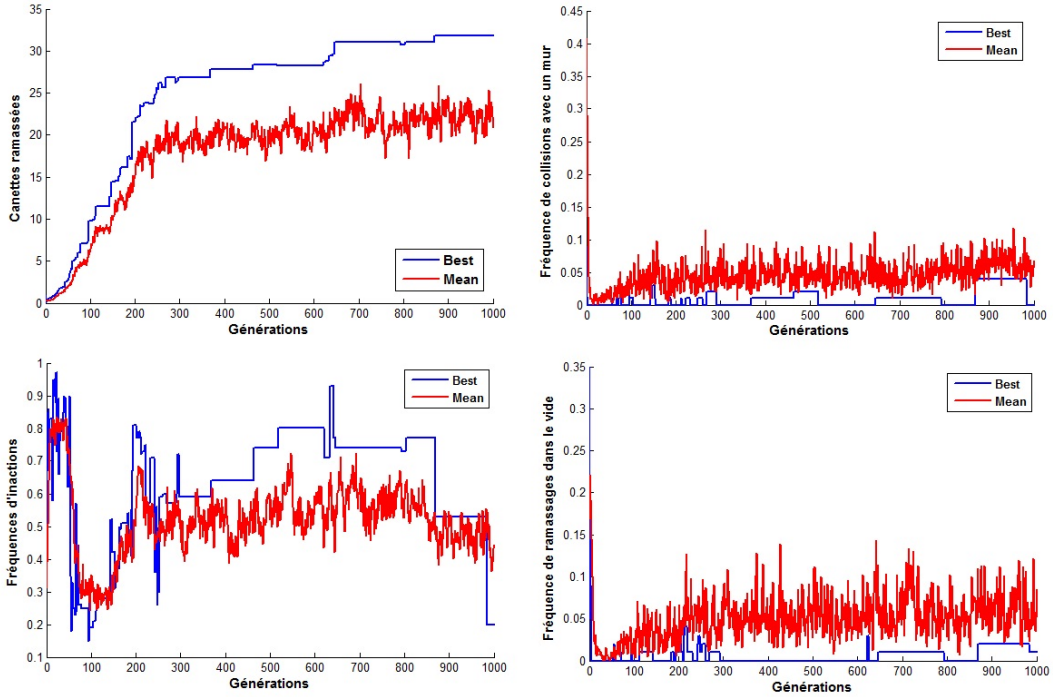


FIGURE 4.9 – Évolution du nombre de canettes ramassées (en haut à gauche), de la fréquence de collision avec un mur (en haut à droite), de la fréquence d'inaction (en bas à gauche) et de la fréquence de ramassage dans le vide (en bas à droite), pour le meilleur individu (en bleu) et pour la moyenne sur la population (en rouge).

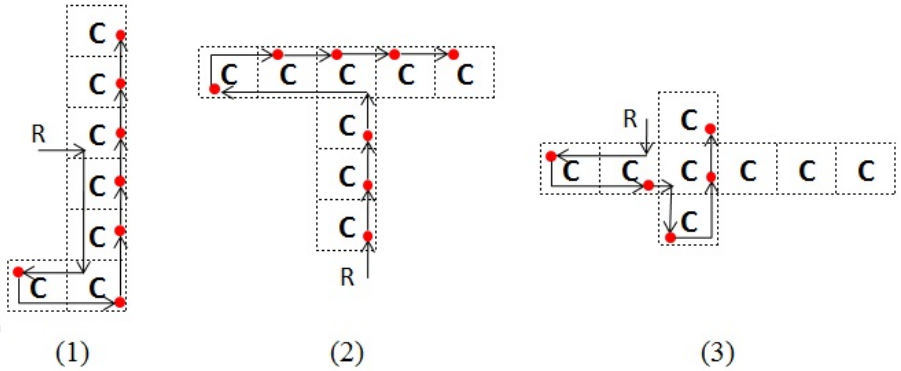


FIGURE 4.10 – Comportement adopté par la stratégie optimale dans trois situations données. Le parcours du robot est détaillé par les flèches, et les points rouges indiquent le ramassage d'une canette.

consiste pas simplement à ramasser une canette à chaque fois que la case courante en contient une. On peut en effet compter que le robot décide d'attraper la canette seulement pour 36 des 64 états (possibles) où la case courante contient une canette. Lorsqu'on affiche la succession des décisions prises par le robot, on se rend compte que celui-ci applique en fait une stratégie qui lui permet de ramasser l'entièreté des canettes se trouvant au sein d'un groupe de canettes adjacentes.

Pour mettre en avant la stratégie adoptée, simulons le comportement du robot pour quelques situations possibles. Les situations considérées et le comportement adopté par le robot sont repris sur la figure 4.10. Sur la première image, le robot se situe initialement à la gauche d'un bloc de canettes adjacentes. Il décide d'y rentrer, et afin de récolter l'ensemble des canettes, il décide d'abord de descendre le plus possible *sans prendre de canette*, une fois en bas, aller le plus à gauche possible toujours sans prendre de canette, et alors, une fois arrivé à l'extrême, il fait tout le chemin inverse en récoltant à chaque étape les canettes se trouvant sur son passage. De manière analogue, sur la deuxième figure, le robot se situe initialement juste en dessous d'un groupe de canettes disposées en "L". Celui-ci décide d'y rentrer et de monter tout en récoltant les canettes. Arrivé à la case la plus au nord, il décide de ne *pas* prendre la canette, mais de d'abord se déplacer le plus à gauche possible pour ensuite faire le chemin inverse en récoltant de nouveau les canettes sur son passage. Rappelons que le robot a une vision limitée, et que, s'il avait décidé de prendre la canette directement sur la case d'intersection des deux lignes de canettes, il aurait dû après faire un choix entre un mouvement à gauche ou à droite, mais sans savoir revenir après pour récolter le reste des canettes. Il aurait donc été impossible de collecter en une fois l'ensemble des canettes en faisant ce choix, et il aurait fallu attendre que le robot revienne à proximité du groupe de canettes restantes pour achever le travail. Enfin, sur la troisième figure, le robot procède de manière analogue aux deux fois précédentes, mais arrivé à l'intersection, au lieu de monter d'abord sans prendre la canette et d'ensuite redescendre en collectant toutes les canettes, il décide de prendre quand même la canette et de poursuivre son chemin vers le nord, rendant impossible la récolte du reste des canettes tant qu'il ne revient pas à proximité. Ces trois figures illustrent bien l'idée de la stratégie pour récolter le plus de canettes possibles lorsque celles-ci sont disposées en groupe : d'abord descendre le plus possible au sud-ouest sans prendre de canette, et ensuite refaire le chemin inverse en récoltant les canettes rencontrées.

Notons cependant que la stratégie trouvée n'est pas *parfaite*. En effet, certains comportements pourraient être améliorés, même si l'idée principale de la stratégie est très bonne. Pour donner des exemples de gènes qui pourraient être améliorés, on peut considérer d'abord l'état où le robot est au-dessus d'une canette, mais ne possède ni de canette sur sa case courante, ni sur

les autres cases avoisinantes. L'action associée est un mouvement aléatoire, alors qu'il semble clair qu'un mouvement vers le sud serait plus performant. De la même manière, lorsque le robot se situe à droite d'une canette, sur une case vide, et juste en dessous du mur au nord, l'action est de réaliser un mouvement aléatoire alors qu'il semblerait plus judicieux de se déplacer vers l'ouest. Cependant, ces exemples n'ont de répercutions que sur le temps total mis par le robot pour trouver l'ensemble des canettes de la grilles. Par contre, certains comportements sont plus problématiques car ils empêchent le robot de ramasser les canettes qui sont dans certains états particuliers, amenant donc forcément à l'impossibilité d'un score parfait de la fitness, et ce, même en augmentant le nombre d'actions de la session. Par exemple, lorsque le robot se situe dans le coin inférieur droit et qu'une canette se trouve dans cette case, et que les cases voisines sont vides, la stratégie trouvée décide d'effectuer un mouvement aléatoire, alors qu'il faudrait absolument prendre la canette. En effet, seuls les états pour lesquels au moins l'une des cases voisines contient une canette peuvent se permettre de ne pas ramasser la canette se trouvant dans la case courante, car cela correspond à l'idée de revenir prendre la canette "plus tard". Pour les états dont les cases avoisinantes sont vides, le fait de "revenir plus tard" ne changera rien et le robot ne ramassera jamais la canette concernée. Pour illustrer ce problème, nous augmentons le nombre d'actions permises pour les sessions de nettoyage. La fitness obtenue avec 1000 actions est de 0.9783, ce qui n'est pas beaucoup plus élevé que la fitness obtenue avec 200 actions. Dans la section suivante nous tentons de déduire la solution optimale à partir de la solution issue du GA.

4.4.4 Déduction d'une solution optimale

Même si le GA ne permet pas d'obtenir l'optimum global (ou du moins pas en un temps raisonnable), nous pouvons définir une stratégie qui respecte l'idée principale de la solution obtenue par le GA, qui consiste à se déplacer à l'une des extrémités dans les groupes de canettes, de manière à pouvoir toutes les ramasser en un seul passage par la suite.

Nous définissons donc une stratégie grâce aux conditions suivantes, basées sur le nombre n_c de canettes détectées par le robot dans son voisinage direct :

- $n_c = 0$: faire un mouvement aléatoire.
- $n_c = 1$: si c'est la case courante qui contient la canette, la ramasser. Sinon, se déplacer dans la direction vers laquelle elle se trouve.
- $n_c = 2$: si la case courante contient une canette, la ramasser. Sinon, se diriger vers la première direction dans laquelle le robot trouve une canette, en considérant l'ordre suivant : (1) sud, (2) ouest, (3) nord et (4) est.

- $n_c = 3$: si la case courante contient une canette, se diriger dans la première direction dans laquelle il y a une canette, en considérant le même ordre qu'avant, *sauf* dans le cas où les cases nord et est contiennent une canette, où il faut alors ramasser la canette. Si la case courante est vide, se diriger vers la première direction dans laquelle il y a une canette, toujours dans l'ordre S,O,N,E.
- $n_c = 4$: si la case courante ne contient *pas* de canette, faire un mouvement aléatoire. Sinon se déplacer selon la même stratégie de la première direction qui contient une canette dans l'ordre de considération donnée.
- $n_c = 5$: prendre la canette qui se situe à la place courante.

Notons que le choix de faire un mouvement aléatoire lorsque $n_c = 0$ est motivé par le fait que de cette manière, la stratégie devrait passer par chaque point de la grille une infinité de fois, et la probabilité de passer par un point donné en un temps fini est de 1 (peut être démontré grâce à la théorie des marches aléatoires). De cette manière, le robot devrait ramasser toutes les canettes, pour n'importe quelle grille, du moment que le nombre d'actions est assez grand. D'autre part, l'exception dans le cas où $n_c = 3$ est indispensable pour éviter que le robot ne tourne en rond dans un bloc. Il faut une extrémité par laquelle il commence à ramasser. Une représentation de la stratégie ainsi formée est donnée sur la figure 4.11.

Calculons la fitness de la stratégie définie par ces conditions. Pour des sessions limitées à 200 actions, la fitness est de 0.9596, ce qui est *moins* que la fitness de la stratégie obtenue par le GA. Cependant, en augmentant le nombre d'actions, la fitness de notre stratégie augmente également. La figure 4.12 représente l'évolution de la fitness (et donc du nombre de canettes ramassées) en fonction du nombre d'actions par session de nettoyage, pour la stratégie obtenue avec le GA et pour la stratégie définie dans cette section. Il n'y a en réalité que pour un nombre d'actions compris entre 120 et 290 environ que la stratégie obtenue par le GA est *légèrement* meilleure que notre définition. Après le seuil de 290 actions, la stratégie que nous avons définie tend vers une valeur de la fitness de 1, et donc tend à ramasser l'entièreté des canettes de la grille, alors que la stratégie issue du GA ne ramasse pas toutes les canettes, même lorsque le nombre d'actions par session est grand. La stratégie définie dans cette section est donc meilleure dans l'absolu, puisqu'elle permet de ramasser toutes les canettes avec probabilité 1. Notons que les courbes de la figure 4.12 ne sont pas "lisses" car la fitness est, rappelons-le, calculée comme la moyenne des récompenses sur $n_s = 100$ sessions, et est donc variable d'une évaluation à l'autre.

Curr.	North	South	West	East	Action
Ø	Ø	Ø	Ø	Ø	????
Ø	Ø	Ø	Ø	C	--->
Ø	Ø	Ø	Ø	|	????
Ø	Ø	Ø	C	Ø	<---
Ø	Ø	Ø	C	C	<---
Ø	Ø	Ø	C	|	<---
Ø	Ø	Ø	|	Ø	????
Ø	Ø	Ø	|	C	--->
Ø	Ø	Ø	|	|	????
Ø	Ø	C	Ø	Ø	vvvv
Ø	Ø	C	Ø	C	vvvv
Ø	Ø	C	Ø	|	vvvv
Ø	Ø	C	C	Ø	vvvv
Ø	Ø	C	C	C	vvvv
Ø	Ø	C	C	|	vvvv
Ø	Ø	C	|	Ø	vvvv
Ø	Ø	C	|	C	vvvv
Ø	Ø	C	|	|	vvvv
Ø	Ø	|	Ø	Ø	????
Ø	Ø	|	Ø	C	--->
Ø	Ø	|	Ø	|	????
Ø	Ø	|	C	Ø	<---
Ø	Ø	|	C	C	<---
Ø	Ø	|	C	|	<---
Ø	Ø	|	|	Ø	????
Ø	Ø	|	|	C	--->
Ø	Ø	|	|	|	????
Ø	C	Ø	Ø	Ø	^^^^
Ø	C	Ø	Ø	C	^^^^
Ø	C	Ø	Ø	|	^^^^
Ø	C	Ø	C	Ø	<---
Ø	C	Ø	C	C	<---
Ø	C	Ø	C	|	<---
Ø	C	Ø	|	Ø	^^^^
Ø	C	Ø	|	C	^^^^
Ø	C	Ø	|	|	^^^^
Ø	C	C	Ø	Ø	vvvv
Ø	C	C	Ø	C	vvvv
Ø	C	C	Ø	|	vvvv
Ø	C	C	C	Ø	vvvv
Ø	C	C	C	C	????
Ø	C	C	C	|	vvvv
Ø	C	C	|	Ø	vvvv
Ø	C	C	|	C	vvvv
Ø	C	C	|	|	vvvv
Ø	C	|	Ø	Ø	^^^^
Ø	C	|	Ø	C	^^^^
Ø	C	|	Ø	|	^^^^
Ø	C	|	C	Ø	<---
Ø	C	|	C	C	<---
Ø	C	|	C	|	<---
Ø	C	|	|	Ø	^^^^
Ø	C	|	|	C	^^^^
Ø	C	|	|	|	^^^^

Curr.	North	South	West	East	Action
Ø	|	Ø	Ø	Ø	????
Ø	|	Ø	Ø	C	--->
Ø	|	Ø	Ø	|	????
Ø	|	Ø	C	Ø	<---
Ø	|	Ø	C	C	<---
Ø	|	Ø	C	|	<---
Ø	|	Ø	|	Ø	????
Ø	|	Ø	|	C	--->
Ø	|	Ø	|	|	????
Ø	|	C	Ø	Ø	vvvv
Ø	|	C	Ø	C	vvvv
Ø	|	C	Ø	|	vvvv
Ø	|	C	C	Ø	vvvv
Ø	|	C	C	C	vvvv
Ø	|	C	C	|	vvvv
Ø	|	C	|	Ø	vvvv
Ø	|	C	|	C	vvvv
Ø	|	C	|	|	vvvv
Ø	|	|	Ø	Ø	????
Ø	|	|	Ø	C	--->
Ø	|	|	Ø	|	????
Ø	|	|	C	Ø	<---
Ø	|	|	C	C	<---
Ø	|	|	C	|	<---
Ø	|	|	|	Ø	????
Ø	|	|	|	C	--->
Ø	|	|	|	|	????
C	Ø	Ø	Ø	Ø	TAKE
C	Ø	Ø	Ø	C	TAKE
C	Ø	Ø	Ø	|	TAKE
C	Ø	Ø	C	Ø	TAKE
C	Ø	Ø	C	C	<---
C	Ø	Ø	C	|	TAKE
C	Ø	Ø	|	Ø	TAKE
C	Ø	Ø	|	C	TAKE
C	Ø	Ø	|	|	TAKE
C	Ø	C	Ø	Ø	TAKE
C	Ø	C	Ø	C	vvvv
C	Ø	C	Ø	|	TAKE
C	Ø	C	C	Ø	vvvv
C	Ø	C	C	C	vvvv
C	Ø	C	C	|	vvvv
C	Ø	C	|	Ø	TAKE
C	Ø	C	|	C	vvvv
C	Ø	C	|	|	TAKE
C	Ø	|	Ø	Ø	TAKE
C	Ø	|	Ø	C	TAKE
C	Ø	|	Ø	|	TAKE
C	Ø	|	C	Ø	TAKE
C	Ø	|	C	C	<---
C	Ø	|	C	|	TAKE
C	Ø	|	|	Ø	TAKE
C	Ø	|	|	C	TAKE
C	Ø	|	|	|	TAKE

Curr.	North	South	West	East	Action
C	C	Ø	Ø	Ø	TAKE
C	C	Ø	Ø	C	TAKE
C	C	Ø	Ø	|	TAKE
C	C	Ø	C	Ø	<---
C	C	Ø	C	C	<---
C	C	Ø	C	|	<---
C	C	Ø	|	Ø	TAKE
C	C	Ø	|	C	TAKE
C	C	Ø	|	|	TAKE
C	C	C	Ø	Ø	vvvv
C	C	C	Ø	C	vvvv
C	C	C	Ø	|	vvvv
C	C	C	C	Ø	vvvv
C	C	C	C	C	TAKE
C	C	C	C	|	vvvv
C	C	C	|	Ø	vvvv
C	C	C	|	C	vvvv
C	C	C	|	|	vvvv
C	C	|	Ø	Ø	TAKE
C	C	|	Ø	C	TAKE
C	C	|	Ø	|	TAKE
C	C	|	C	Ø	TAKE
C	C	|	C	C	<---
C	C	|	C	|	TAKE
C	C	|	|	Ø	TAKE
C	C	|	|	C	TAKE
C	C	|	|	|	TAKE
C	|	Ø	Ø	Ø	TAKE
C	|	Ø	Ø	C	TAKE
C	|	Ø	Ø	|	TAKE
C	|	Ø	C	Ø	TAKE
C	|	Ø	C	C	<---
C	|	Ø	C	|	TAKE
C	|	Ø	|	Ø	TAKE
C	|	Ø	|	C	TAKE
C	|	Ø	|	|	TAKE
C	|	C	Ø	Ø	TAKE
C	|	C	Ø	C	vvvv
C	|	C	Ø	|	TAKE
C	|	C	C	Ø	vvvv
C	|	C	C	C	vvvv
C	|	C	C	|	vvvv
C	|	C	|	Ø	TAKE
C	|	C	|	C	vvvv
C	|	C	|	|	TAKE
C	|	|	Ø	Ø	TAKE
C	|	|	Ø	C	TAKE
C	|	|	Ø	|	TAKE
C	|	|	C	Ø	TAKE
C	|	|	C	C	<---
C	|	|	C	|	TAKE
C	|	|	|	Ø	TAKE
C	|	|	|	C	TAKE
C	|	|	|	|	TAKE

FIGURE 4.11 – Représentation de la stratégie définie à partir de la solution obtenue grâce au GA.

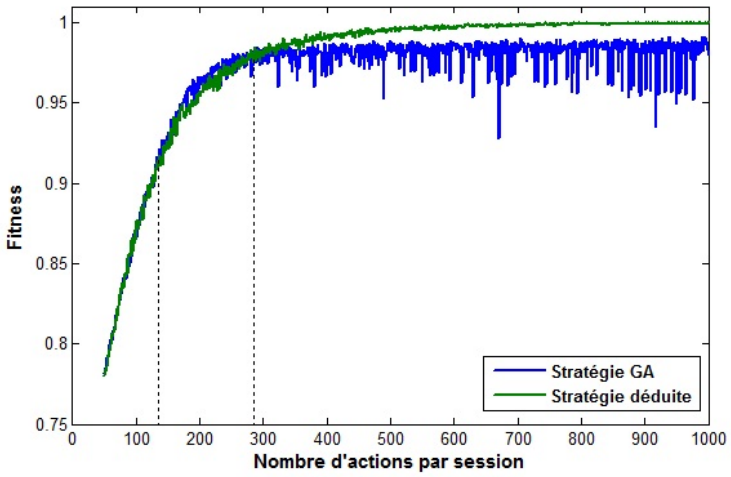


FIGURE 4.12 – Fitness en fonction du nombre d'actions par session de nettoyage, pour la stratégie optimale obtenue avec le GA (bleu) et pour la stratégie déduite (vert).

4.5 Interface graphique

Nous tenons à signaler qu'une interface graphique a été créée pour permettre une manipulation plus facile du GA. Nous présentons brièvement les possibilités de fonctionnement de celle-ci. Pour ce faire, donnons une image "print-screen" de l'interface graphique sur la figure 4.13.

Nous avons scindé l'interface en deux parties. À gauche l'utilisateur a la possibilité de visualiser la simulation du nettoyage d'une session par le robot pour 3 stratégies différentes : (1) une stratégie aléatoire, (2) la meilleure stratégie trouvée par le GA après exécution de la partie de droite et (3) la stratégie optimale mise en avant dans la section 4.4.4. Dans la partie de droite, l'utilisateur peut lancer l'algorithme génétique avec des paramètres qu'il aura lui-même fixé. La figure se situant au milieu de la partie de droite permet de visualiser en temps réel l'évolution de la fitness au cours des générations. Nous affichons aussi la fitness de chaque individu de la population de chaque génération, et affichons la définition de la meilleure stratégie de la génération courante.

Notons que l'interface graphique est pratique pour celui qui voudrait tester les différentes possibilités du GA sans devoir se "plonger dans le code", mais est limitée en terme d'utilisation et de flexibilité. Pour une utilisation plus poussée, il est préférable de se référer au code directement.

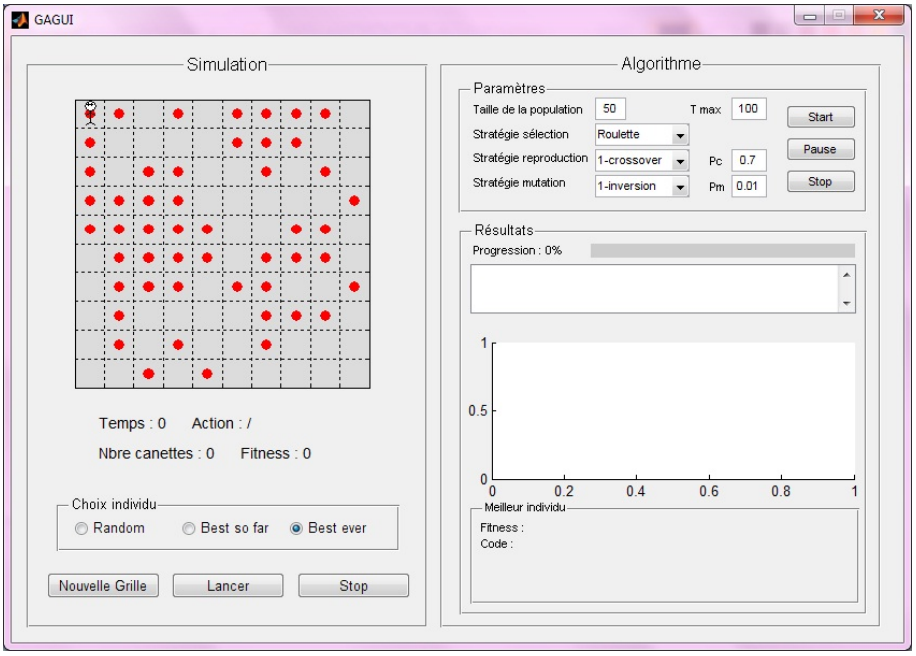


FIGURE 4.13 – Interface graphique pour l'algorithme génétique.

4.6 Conclusion

Ce chapitre a permis d'illustrer le fonctionnement d'un GA sur un problème plus conséquent que l'exemple basique considéré dans le chapitre 2. Cela a permis en particulier de mettre en avant les différentes étapes clés lors de l'application d'un GA.

Tout d'abord, nous avons abordé la question de la modélisation du problème et du codage des individus. Soulignons qu'il a fallu à cette étape faire certains *choix*, tant au niveau de la définition de la fonction objectif qu'au niveau du codage. Il serait intéressant de comparer les résultats obtenus avec d'autres définitions, ou d'autres codages.

Ensuite, nous avons défini la structure du GA, et les différents opérateurs considérés. De nouveau, certains choix ont été fait, dans la structure de l'algorithme et dans les choix d'implémentation. Nous avons alors été confronté au problème du réglage des paramètres du GA. Nous avons vu en effet qu'une bonne paramétrisation était primordiale pour de bonnes performances du GA. Il est donc important d'avoir un ajustement des composants de l'algorithme en fonction des résultats obtenus, et de ne pas appliquer le GA de manière "aveugle".

Enfin, nous avons présenté les résultats obtenus à l'issue de l'exécution du GA correctement paramétré. Celui-ci a permis le développement d'une stratégie intelligente pour le ramassage des canettes, et ce sans utiliser aucune information par rapport à la structure du problème mais uniquement grâce au processus d'évolution du GA. Après analyse du résultat obtenu, nous nous sommes rendu compte que la solution trouvée n'était pas totalement optimale, et qu'elle pouvait encore être améliorée. Toutefois, l'étude de cette solution a permis de dégager l'origine de sa qualité, et de définir grâce à cela une solution qui, elle, était optimale. Cela illustre l'importance de la phase d'analyse des résultats.

Remarquons que l'algorithme génétique pourrait encore être amélioré, et que d'autres analyses intéressantes pourraient être effectuées, tant au niveau du GA que du problème du ramassage de canettes. En effet, une solution optimale étant connue, il est sans doute possible d'inclure dans le GA certaines informations sur la structure problème.

Clôturons cette section en donnant "en vrac" quelques pistes et idées pour améliorer le GA :

- Étudier plus en profondeur l'influence des paramètres du GA sur les performances du GA.
- Pour obtenir une stratégie qui permette de ramasser l'entièreté des canettes présentes sur la grille, on pourrait augmenter le nombre d'actions permises lors des sessions de nettoyage après la première phase de convergence de l'algorithme.
- On pourrait coupler le GA avec des méthodes de recherche locale lorsque celui-ci stagne depuis longtemps. En effet, il semblerait qu'après la première phase de convergence, la stratégie trouvée contienne la structure générale de la solution optimale, mais possède encore quelques gènes défectueux. Une recherche locale permettrait peut être de mieux cibler ceux-ci.
- Comme nous avons vu que la convergence de l'algorithme dépendait de la dispersion de la population, une idée serait d'essayer de contrôler l'algorithme de manière à ce que celle-ci reste dans un intervalle de valeurs qui mène à une bonne convergence. On pourrait par exemple définir le taux, ou même l'opérateur de mutation, en fonction de la dispersion observée dans la population.
- Il serait sans doute intéressant d'étudier aussi les différentes phases d'évolution des stratégies suivant l'évolution de la fitness au cours des générations. Par exemple analyser si le GA permet d'abord d'éviter les murs, ensuite d'éviter de prendre des canettes là où il n'y a rien, et ainsi de suite, ou si l'apprentissage est plutôt couplé et se fait de manière simultané.

Références pour le chapitre 4

Comme nous l'avons déjà mentionné auparavant, les principales sources de ce chapitre sont le livre [Mitchell, 2009], pour la présentation du problème, et les notes de cours [Leclercq, 2010], pour l'étude des paramètres. Remarquons toutefois que seule la présentation du problème et la définition des fonctions objectif et de codage sont reprises de [Mitchell, 2009]. Pour le reste, nous avons voulu "jouer le jeu" en ne s'inspirant pas de la structure algorithmique proposée ou des solutions obtenues. Nous avions uniquement connaissance de l'existence d'une "stratégie particulière" qui permette de récolter un grand nombre de canettes, plus efficacement qu'en ramassant directement les canettes qui se situent sur la case courante, mais sans en connaître la définition exacte.

Chapitre 5

Application des GAs à l'imagerie médicale

Ce chapitre est consacré à l'application des GAs au problème d'optimisation d'expériences pour l'imagerie par résonance magnétique fonctionnelle (fonctional Magnetic Resonance Imaging, fMRI). Celui-ci a été réalisé dans le cadre du stage du master 2 en mathématiques à finalité approfondie, à l'université de Warwick (Royaume-Uni) sous la supervision de Thomas Nichols.

La structure de ce chapitre se présente de la manière suivante. Tout d'abord, nous introduirons les concepts liés à l'imagerie médicale fonctionnelle dans la section 5.1. Ensuite, dans la section 5.2, nous donnerons les deux grands types d'expériences : les expériences par blocs et les expériences par événements. Une fois ces concepts établis, nous passerons à la phase de modélisation des données par le modèle linéaire général (General Linear Model, GLM) et nous expliquerons comment cette modélisation permet de réaliser des inférences sur les données dans la section 5.3. Dans la section 5.4, nous définirons les critères relatifs à l'optimisation d'expériences pour le fMRI, en particulier les critères d'*efficacité*, de *contrebalancement*, du *respect des fréquences* et enfin de la *robustesse*. Ensuite nous expliquerons comment cette optimisation peut être effectuée grâce aux GAs. Finalement, dans la section 5.6 nous présenterons les résultats des simulations numériques effectuées.

Notons que la partie réellement innovatrice de ce chapitre est la considération de la robustesse dans l'optimisation multi-objectifs. En effet, ce critère n'a encore jamais été considéré et l'objectif principal du stage a été d'ajouter ce critère à ceux déjà considérés dans des travaux antérieurs, et d'analyser les résultats obtenus.

5.1 Imagerie par résonance magnétique fonctionnelle (fMRI)

5.1.1 Généralités

L'imagerie par résonance magnétique fonctionnelle (fMRI) est une technique d'imagerie médicale issue de l'imagerie par résonance magnétique (Magnetic Resonance Imaging, MRI) permettant de visualiser indirectement l'activité cérébrale et d'étudier les régions fonctionnelles du cerveau humain.

Lors d'une session fMRI, le sujet repose dans un scanner et un champ magnétique est généré pendant que le sujet réalise un ensemble de *tâches*, dont l'agencement est dicté par l'*expérience* de la session. Par exemple, une expérience pourrait être d'alterner d'une part le tapotement des doigts pendant 30 secondes et d'autre part un repos de 30 secondes, et ce pendant toute la durée de la session.

Durant la session, un signal fMRI est mesuré à des intervalles de temps réguliers TR (Temporal Resolution) en chaque voxel du cerveau (un voxel est un pixel 3D, typiquement de l'ordre de 3mm x 3mm x 3mm). Ce signal se base sur le fait que lorsqu'une région du cerveau est sollicitée, on remarque une augmentation du flux sanguin et de l'oxygénation du sang dans cette région ([Bandetti and Cox, 2001]), et que celle-ci peut être mesurée grâce aux propriétés magnétiques du sang. Ce signal, appelé signal *BOLD* (Blood Oxygen Level Dependant), représente donc une observation *indirecte* de l'activité cérébrale ([Logothetis and Wandell, 2004]). Pour chaque voxel, le signal BOLD peut donc être vu comme la réponse $y(.)$ d'un système dynamique dont l'entrée $s(.)$ est l'expérience effectuée sur le sujet et l'état $x(.)$ est l'activité cérébrale en ce voxel. La représentation d'un tel système est faite sur la figure 5.1.

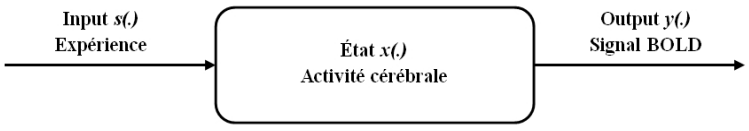


FIGURE 5.1 – Représentation du système dynamique pour l'activité cérébrale.

On obtient donc à l'issue de la session une *série temporelle* $(Y_i)_{i=1}^n$ pour chaque voxel, où n est le nombre de données temporelles obtenues à l'issue du scan, et où Y_i représente la valeur du signal BOLD $y(.)$, supposé continu, au temps $i \cdot \text{TR}$. Une manière équivalente de voir ces données est de les considérer comme étant un échantillon de n observations multivariées, chacune contenant autant d'éléments que de voxels (environ 10^5 pour l'en-

semble du cerveau). De cette manière, on peut effectuer des tests statistiques sur ces données multivariées et par exemple mettre en avant des relations entre régions. Cependant, dans la suite de ce travail nous nous placerons dans un cadre d'étude voxel par voxel, et porterons notre étude sur les séries temporelles en chacun de ceux-ci.

5.1.2 Analyse des données fMRI

L'analyse des données obtenues grâce au fMRI est une tâche rendue complexe suite à plusieurs facteurs. Tout d'abord, les données sont bruitées par les perturbations externes, comme par exemple les mouvements de la tête (et ce malgré la structure qui permet de bloquer la tête). Ensuite, il y a plusieurs sources de variabilité dans les données, notamment les différences entre individus, entre voxels, et même entre différentes sessions sur un même sujet. Enfin, la grande dimension des données nécessite une attention particulière à toutes les méthodes qui manipulent ces données, pour éviter d'une part l'accumulation des erreurs numériques et d'autre part pour éviter d'avoir des programmes trop gourmands en temps de calcul. L'analyse des données issues du fMRI nécessite donc le traitement de chacune de ces considérations. Les principales composantes d'une analyse de données fMRI sont :

- *Contrôle de la qualité* : s'assurer que les données ne sont pas biaisées par des éléments extérieurs.
- *Correction des déformations* : corriger les déformations spatiales qui surgissent souvent dans les données fMRI.
- *Correction des mouvements de la tête* : réaligner les images obtenues pour corriger les effets dûs aux mouvements résiduels de la tête.
- *Correction de l'acquisition des données par tranches* : corriger les erreurs due à l'acquisition des données qui se fait à des temps différents selon les différentes parties du cerveau. En effet, le fMRI ne permet que d'obtenir des tranches d'environ 64 x 64 voxels par scan. Il faut donc plusieurs scans pour obtenir les données pour le cerveau dans son entièreté, et les différentes tranches ont des temps d'acquisition légèrement différents.
- *Normalisation spaciale* : aligner les données dans un espace commun (un "cerveau de référence") de manière à pouvoir comparer les données provenant d'individus différents.
- *Lissage spatial* : lisser les données pour réduire le bruit résiduel des données et obtenir une continuité entre voxels.
- *Filtre temporel* : filtrer les données pour enlever le bruit à basse fréquence dû à l'autocorrélation temporelle des données.
- *Modélisation statistique* : ajuster les données à un modèle statistique de manière à estimer la réponse à une tâche ou à un stimulus.

- *Inférence statistique* : estimer le taux de signification des résultats de la modélisation statistique, en tenant compte du grand nombre de tests statistiques effectués dans l'ensemble du cerveau.
- *Visualisation* : permettre une visualisation des données grâce à des outils graphiques.

Nous ne nous intéresserons ici qu'à la partie de modélisation statistique (ainsi qu'aux conséquences de notre étude sur l'inférence statistique). Nous supposerons donc pour la suite être en possession de données ayant déjà subies toutes les corrections citées avant. Notons tout de même que malgré ces corrections, les données resteront plus ou moins bruitées, ce que nous devrons prendre en compte dans la phase de modélisation (cf. section 5.3.1).

5.1.3 Système linéaire à temps invariant (LTI)

Considérons le signal BOLD pour un voxel donné. Comme nous l'avons déjà mentionné dans la section 5.1.1, celui-ci peut être vu comme la réponse du système dont l'entrée est l'expérience menée sur le sujet et l'état est l'activité cérébrale. Certains travaux mettent en avant le fait que ce système peut être considéré comme *linéaire* dans certains cas. En effet, la référence [Dale and Buckner, 1997] présente une expérience où des stimuli sont présentés successivement de manière assez rapide, et montre que la réponse prédite en supposant la linéarité du système, *i.e.* en sommant linéairement les réponses propres à chaque stimulus pris séparément, colle "assez bien" aux données observées, et donc que le système peut être considéré comme linéaire.

Cependant, dans certains cas, on peut observer des non-linéarités. C'est le cas notamment lorsqu'un stimulus est présenté trop longtemps, car alors le sujet démontre une accoutumance et le signal décroît progressivement jusqu'à une certaine valeur. De même, on peut également observer une *saturation* du signal BOLD lorsque les stimuli sont présentés de manière trop rapide (moins de 2 secondes, [Wager et al., 2005]). Enfin, une autre source de non-linéarité a été mise en avant dans [Yeşilyurt et al., 2008], où il est montré que la réponse à un stimulus visuel d'une durée de 5 millisecondes amène à une réponse d'amplitude seulement deux fois plus petite que celle engendrée par un stimulus de 1 seconde.

Malgré le fait évident que dans certains cas les non-linéarités du système sont très importantes, la plupart des études se placent dans le cadre où il peut être fait l'hypothèse de manière raisonnable que le système étudié est linéaire. De manière plus générale, on peut supposer être dans le cas d'un système linéaire à temps-invariant (Linear Time Invariant, LTI), c'est-à-dire faire l'hypothèse supplémentaire que la réponse à une entrée donnée ne varie pas suivant le moment particulier où celle-ci est effectuée. Dès lors, il est

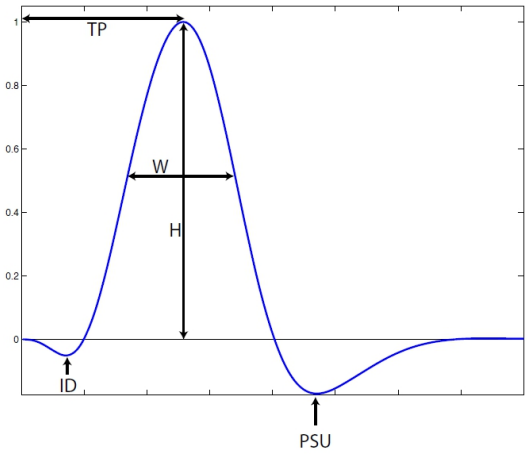


FIGURE 5.2 – Forme typique pour la réponse hémodynamique HRF. Source : [Poldrack et al., 2010]

possible d'exprimer la réponse du système comme la *convolution* entre la fonction d'entrée et la *réponse impulsionnelle*, appelée aussi dans le contexte du fMRI *réponse hémodynamique* (Hemodynamic Response Function, HRF), qui correspond à la réponse provoquée par un bref stimulus (plus formellement par une entrée correspondant à la distribution de Dirac δ). Étant donné une fonction d'entrée $s(.)$ et la réponse impulsionnelle $h(.)$, l'opérateur de convolution, noté $\otimes$, est donné par

$$(s \otimes h)(t) := \int_{-\infty}^{+\infty} s(\tau)h(t-\tau)d\tau. \tag{5.1}$$

Si la fonction HRF est connue, l'opérateur de convolution permet donc de *prédire* la réponse du système pour n'importe quelle entrée donnée. La forme typique de la réponse impulsionnelle pour le signal BOLD est donnée sur la figure 5.2. Plusieurs caractéristiques de cette fonction varient d'un individu à l'autre, et d'un voxel à l'autre au sein d'un même individu, citons notamment la *durée au sommet* TP, la *largeur* W, l'*amplitude* H, la *décroissance initiale* ID, et la *décroissance post-stimulus* PSU. La représentation de ces paramètres est également donnée sur la figure 5.2.

Pour terminer cette section, illustrons l'effet de l'opérateur de convolution pour les systèmes LTI. Pour cela considérons deux fonctions d'entrées : la première est une illustration de l'exemple donné dans la section 5.1.1 avec l'alternance de phases où un stimulus est appliqué de manière continue et de phases de repos, le deuxième est une expérience où l'on considère une suite "désordonnée" de stimulus *brefs* (de type $\delta(.)$). La représentation de ces fonctions d'entrée et des réponses après convolution est faite sur la figure 5.3.

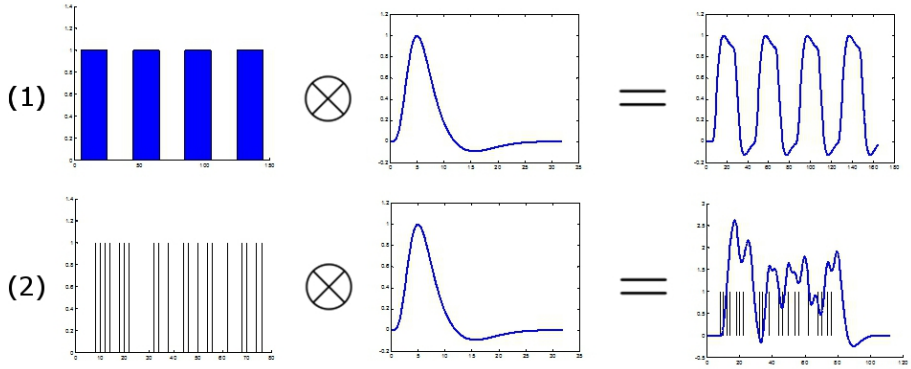


FIGURE 5.3 – Exemples de convolution pour un système LTI pour (1) des stimuli par blocs et (2) une suite désordonnée de stimuli de type δ.

5.2 Types d'expériences pour le fMRI

Nous allons nous pencher maintenant sur les expériences accomplies sur le sujet lors d'une session fMRI, c'est-à-dire la fonction d'entrée $s(.)$ du système lié au signal BOLD. De manière générale, on différentie les *types* de stimuli appliqués (par exemple les stimuli visuels, les stimuli auditifs, les stimuli moteurs, etc.), et on définit la fonction d'entrée comme

$$s(.) = (s_i(.))_{i=1}^{Q}, \tag{5.2}$$

où Q est le nombre de stimuli différents et $s_i(.)$ représente la fonction d'entrée pour les stimuli de type i, pour $i = 1, \ldots, Q$.

Dans les premières études fMRI, les expériences étaient menées grâce à l'alternance de longues périodes de stimuli soumis de manière continue, comme l'exemple de la section 5.1.1, avec éventuellement plus d'un type de stimulus. Cette méthode était typique de la tomographie par émission de positrons (Positron Emission Tomography, PET), une autre méthode de neuroimagerie inventée dans les années 1980. L'analyse des données obtenues consistait alors à prendre la moyenne du signal obtenu pour chaque type de stimulus au sein d'une période et de réaliser des inférences grâce à de simples tests t ([Buckner et al., 1996]). Cependant, le fMRI ayant une précision temporelle et spatiale beaucoup grande que le PET (de 1.5 à 6 secondes contre 2 minutes et de 15 à 60 mm^3 contre 250 à 1000 mm^3), il est possible de considérer d'autres formes de fonctions d'entrée, et en particulier les fonctions formées par une succession d'*impulsions* (*i.e.* de stimuli très brefs) car la précision du fMRI permet de mesurer la réponse à un impulsion isolée ([Buckner et al., 1996]). Les chercheurs ont donc commencé à étudier des expériences dites *par événements*, utilisant des impulsions de stimuli plutôt que des périodes de présentation continues. La seule contrainte est de ne

pas trop rapprocher deux stimuli successifs, car alors il devient impossible de distinguer leur effet respectif : l'intervalle inter-stimulus (Inter-Stimulus Interval, ISI) est choisi entre 2 et 20 s. On peut alors envisager de considérer des expériences où l'ordre des stimuli et l'ISI sont déterminés de manière aléatoire. Une illustration d'un tel type d'expérience est faite sur la figure 5.4.

Les entrées $s_i(.)$ considérées dans ce travail seront donc de la forme

$$s_i(t) = \sum_{j=1}^{n_i} \delta(t - t_{ij}), \tag{5.3}$$

où n_i est le nombre de stimuli de type i, t_{ij} représente le temps où le j^e stimulus de type i est appliqué, et où $\delta(. - t_{ij})$ désigne la distribution de Dirac décalée au temps t_{ij}. De plus, comme la distribution de Dirac vérifie

$$\forall t_{ij},\ (\delta(. - t_{ij}) \otimes h(.))(t) = h(t - t_{ij}), \tag{5.4}$$

la fonction de réponse y_i à la i^e entrée sera donnée par

$$y_i(t) = \sum_{j=1}^{n_i} h(t - t_{ij}). \tag{5.5}$$

Remarquons que cette définition n'empêche pas de considérer quand même des expériences par blocs, il suffit de faire des groupes de stimuli de même type en choisissant un ISI de longueur minimum et constant, comme montré à la figure 5.4.

Pour les expériences considérées dans ce travail, nous supposerons que l'ISI est constant et égal au temps inter-scans TR. Nous aurons donc un événement (éventuellement un repos) toutes les TR secondes. Même si cela est un peu plus restrictif que la définition donnée par l'équation (5.3), le fait d'inclure des événements nuls permet d'avoir des ISI (entre les événements non nuls) non constants, mais avec la contrainte de devoir être un multiple de TR. L'ensemble des solutions admissibles correspond donc à l'ensemble des expériences à temps ISI=TR constants pour une session de durée $t = n \cdot \mathrm{TR}$ secondes, où n est le nombre de données acquises à l'issue du scan.

5.3 Modélisation

Considérons la série temporelle $(Y_i)_{i=1}^n$ pour un voxel donné, où n désigne le nombre de données temporelles obtenues à l'issue de la session. Notons également Q le nombre de types de stimuli différents. Pour pouvoir répondre à certaines questions statistiques (par exemple : "*Quels sont les stimuli qui agissent sur le voxel considéré ?*"), il est pratique d'ajuster les données à un

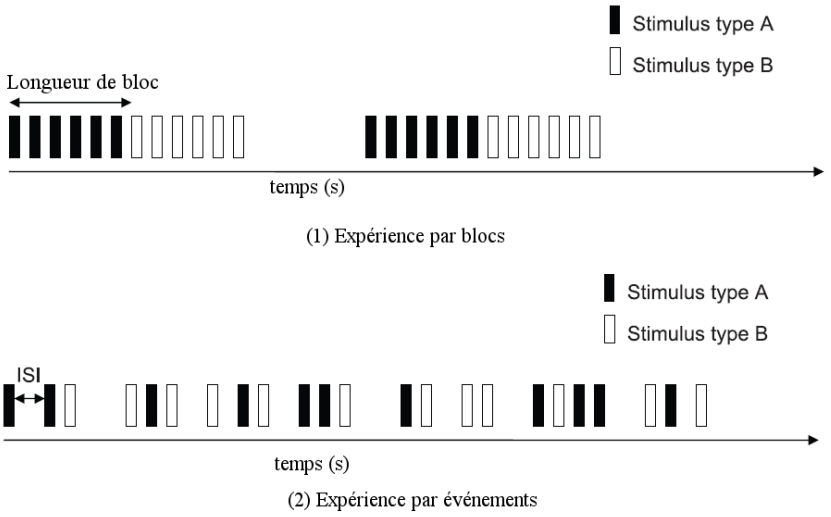


FIGURE 5.4 – Expériences par blocs (1) et par événements (2) pour deux types de stimulus. Source : [Maus, 2011].

modèle statistique : le modèle linéaire général (General Linear Model, GLM). La section 5.3.1 donne une présentation succincte de ce modèle. Ensuite nous considérerons l'application de ce modèle pour la *détection* de l'activité cérébrale dans la section 5.3.2. Enfin, nous verrons comment une inférence statistique est possible à partir de ce modèle.

5.3.1 Modèle Linéaire Général (GLM)

Le GLM est utilisé dans beaucoup d'analyses fMRI. Comme son nom l'indique, c'est un outil très général, qui permet par exemple de réaliser (dans un cadre tout à fait quelconque) des régressions linéaires, des tests de corrélations, des tests t ou encore des analyses de variance (ANOVA) ou de covariance (ANCOVA).

Supposons avoir n variables aléatoires $Y_1, \ldots, Y_n$ dont les espérances respectives peuvent s'exprimer comme combinaison linéaire de Q quantités connues $x_{i1}, \ldots, x_{iK}$, *i.e.* qu'il existe Q coefficients $\beta_1, \ldots, \beta_Q$ tels que

$$E(Y_i|x) = \sum_{j=1}^{Q} \beta_j x_{ij}, \ \forall i = 1, \ldots, n. \tag{5.6}$$

Supposons aussi que la variance de ces variables vaut σ^2. Cela peut se noter

$$Y_i \sim \mathcal{N}(\sum_{j=1}^{Q} \beta_j x_{ij}, \sigma^2), \tag{5.7}$$

ou de manière équivalente

$$Y_i = \sum_{j=1}^{Q} \beta_j x_{ij} + \epsilon_i;\ \epsilon_i \sim \mathcal{N}(0, \sigma^2). \tag{5.8}$$

Notons maintenant Σ la matrice de corrélation des variables Y_i, c'est-à-dire définissons

$$\Sigma_{ij} := Cor(\epsilon_i, \epsilon_j). \tag{5.9}$$

En posant

$$Y := (Y_1, \ldots, Y_n)^n, \tag{5.10}$$

et

$$X := \begin{pmatrix} x_{11} & \cdots & x_{1K} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nK} \end{pmatrix}, \tag{5.11}$$

on peut exprimer les équations (5.8) de manière vectorielle par

$$Y = X\beta + \epsilon;\ \epsilon \sim \mathcal{N}(0, \sigma^2\Sigma), \tag{5.12}$$

où la notation $\epsilon \sim \mathcal{N}(0, \sigma^2\Sigma)$ signifie que chaque composante ϵ_i est normalement distribuée avec moyenne nulle et variance $\sigma^2\Sigma_{ii}$. La covariance de l'erreur est quant à elle donnée par les éléments non-diagonaux de $\sigma^2\Sigma$. Plus précisément, on a que $Cov(\epsilon_i, \epsilon_j) = \sigma^2\Sigma_{ij}$. La matrice $\sigma^2\Sigma$ est appelée matrice de *variance-covariance* du vecteur aléatoire ϵ.

Lorsque les coefficients $\beta_1, \ldots, \beta_Q$ de la combinaison linéaire (5.6) sont inconnus, une manière d'estimer leur valeur est d'utiliser la méthode des *moindres carrés*, qui minimise le carré des distances entre les observations et prédictions. La méthode des moindres carrés peut être formulée par le problème de minimisation suivant :

$$\min_{\beta} \|Y - X\beta\|^2 = \min_{\beta} \sum_{i=1}^{n} (Y_i - \sum_{j=1}^{Q} x_{ij}\beta_j)^2, \tag{5.13}$$

La solution de ce problème, lorsque la matrice X est de rang plein, est donné par

$$\hat{\beta} = (X^T X)^{-1} X^T Y. \tag{5.14}$$

Nous avons noté $\hat{\beta}$ pour dénoter le fait qu'il s'agit d'un *estimateur*, et donc d'une *variable aléatoire*. Il s'agit d'ailleurs d'un estimateur *non-biaisé* de β. En effet :

$$\begin{aligned} E(\hat{\beta}) &= E((X^T X)^{-1} X^T Y) \\ &= (X^T X)^{-1} X^T E(Y) \\ &= (X^T X)^{-1} X^T X \beta \\ &= \beta. \end{aligned} \tag{5.15}$$

Sous l'hypothèse supplémentaire que $\Sigma = I$ est la matrice identité, c'est-à-dire que les données sont *indépendantes* entre elles, l'estimateur $\hat{\beta}$ défini par la solution des moindres carrés est l'estimateur ayant la plus petite variance parmi l'ensemble des estimateurs non-biaisés de β (peut être montré grâce au théorème de Gauss Markov, [Graybill, 1961]).

Donnons maintenant un exemple d'application du GLM : la régression linéaire. Supposons être en possession de T données $y_1, \ldots, y_T$ de la mesure d'une variable quantitative pour T individus dont une variable indépendante prend des valeurs respectives de $x_1, \ldots, x_T$. On veut déterminer la relation linéaire entre les deux variables, c'est-à-dire déterminer l'équation de la droite qui passe le mieux par les points (x_i, y_i). Une telle droite a pour équation

$$y = \beta_0 + \beta_1 x, \tag{5.16}$$

où β_0 est l'*ordonnée à l'origine* et β_1 est le *coefficient angulaire* de la droite.

Cette équation peut être réécrite de manière équivalente sous la forme (5.12) en définissant

$$\beta = (\beta_0, \beta_1)^T \tag{5.17}$$

et

$$X = \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}. \tag{5.18}$$

Dans cette formulation, l'erreur ϵ_i est la différence entre l'observation y_i et la valeur $y = \beta_0 + \beta_1 x_i$ prise sur la droite en $X = x_i$. La solution des moindres carrés donne donc dans ce cas les coefficients de la droite (5.16) tels que le carré des distances verticales entre la droite et les observations (x_i, y_i) soit minimum. La représentation d'une telle régression ainsi que des paramètres β_0 et β_1 est faite à la figure 5.5.

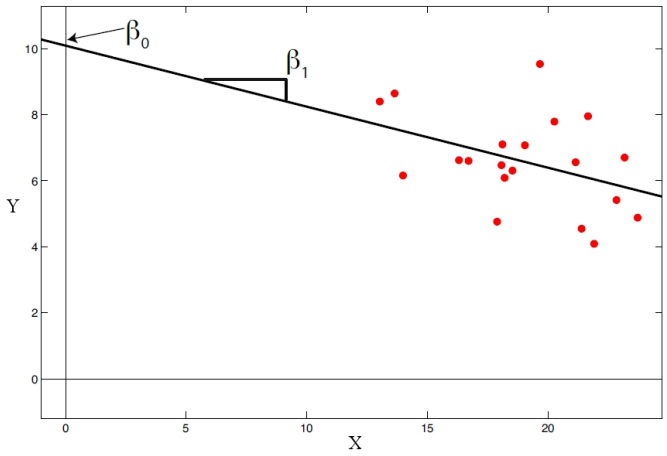


FIGURE 5.5 – Exemple de l'application du GLM pour la régression linéaire.

5.3.2 Détection

Revenons maintenant au contexte de données fMRI. Le GLM peut être appliqué au problème de *détection*, qui est le fait d'estimer les *amplitudes* des fonctions HRF pour les différents types de stimuli, en supposant fixée la forme de la fonction HRF. L'amplitude estimée est alors une mesure de l'activation du voxel considéré pour un type de stimulus donné.

La fonction HRF utilisée est souvent prise comme une différence de deux fonctions gamma ([Maus, 2011], pp. 11) :

$$h(t,(a_1,b_1,c,a_2,b_2)) = \frac{b_1^{a_1+1}t^{a_1}\exp(-b_1 t)}{\Gamma(a_1+1)} - \frac{b_2^{a_2+1}t^{a_2}\exp(-b_2 t)}{c\Gamma(a_2+1)}, \quad (5.19)$$

avec les paramètres a_1, b_1, c, a_2 et b_2 fixés aux valeurs $(a_1,b_1,c,a_2,b_2) = (5,1,6,15,1)$. Cette fonction est appelée la fonction HRF *canonique*, et est notée h_c. C'est cette définition qui est utilisée pour la figure 5.3.

Le but du modèle de détection est donc d'estimer les amplitudes des réponses impulsionnelles pour chaque type de stimulus. Pour ce faire, on exprime le signal BOLD $y(.)$ comme une combinaison linéaire des réponses prédites grâce à l'opérateur de convolution en considérant la fonction HRF h_c pour tous les types de stimulus, plus un terme d'erreur ϵ :

$$y(t) = \sum_{j=1}^{Q} \beta_j \underbrace{(s_j \otimes h_c)}_{=:x_j}(t) + \epsilon = \sum_{j=1}^{Q} \beta_j x_j(t) + \epsilon. \quad (5.20)$$

En posant $y_i := y(i \cdot \mathrm{TR})$ et $X_{i,j} := x_j(i \cdot \mathrm{TR})$, on peut écrire le problème de détection sous la forme du GLM, *i.e.* $Y = X\beta + \epsilon$.

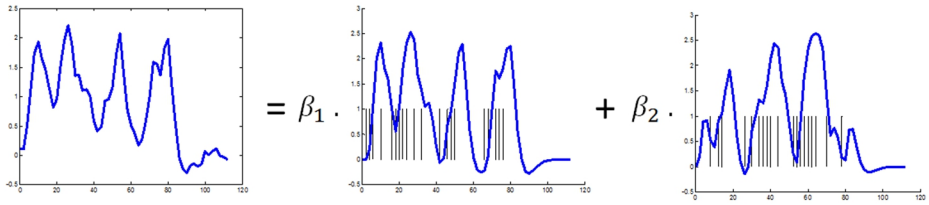


FIGURE 5.6 – Illustration du modèle de détection pour deux types de stimuli.

Le vecteur Y correspond en fait au vecteur des données obtenues à la résolution temporelle TR, et les colonnes de la matrice X correspondent aux prédictions de la réponse du signal BOLD pour chaque type de stimulus, discrétisées à la résolution temporelle TR. L'illustration du modèle de détection pour deux types de stimuli est faite sur la figure 5.6.

Pour ce modèle, l'erreur ϵ désigne le bruit résiduel après le pré-traitement des données (cf. section 5.1.2). L'hypothèse $\epsilon \sim \mathcal{N}(0, \sigma^2\Sigma)$ signifie donc que l'espérance de ce bruit résiduel est nulle, et que la corrélation entre les données est décrite par la matrice Σ. Comme les données représentent la série temporelle d'un signal, la corrélation est en fait la corrélation *temporelle* qu'il existe dans les données. Une méthode pour corriger cette corrélation temporelle est de prémultiplier chaque coté du GLM par la matrice K de décomposition de Cholesky de Σ^{-1}, vérifiant $\Sigma^{-1} = K^T K$ ([Friston et al., 1999], [Friston et al., 2000]). Le nouveau système devient alors

$$KY = KX\beta + K\epsilon, \tag{5.21}$$

et l'erreur $K\epsilon$ est indépendante. En effet,

$$Cor(K\epsilon) = KCor(\epsilon)K^T = K(K^TK)^{-1}K^T = KK^{-1}K^{-T}K^T = I. \tag{5.22}$$

En posant alors $\tilde{Y} := KY$, $\tilde{X} := KX$ et $\tilde{\epsilon} := K\epsilon$, on peut réécrire (5.21) comme

$$\tilde{Y} = \tilde{X}\beta + \tilde{\epsilon}. \tag{5.23}$$

L'estimateur du paramètre β est alors donné par

$$\hat{\beta} = (\tilde{X}^T\tilde{X})^{-1}\tilde{X}^T\tilde{Y} = (X^T\Sigma X)^{-1}X^T\Sigma^{-1}Y, \tag{5.24}$$

et sa distribution est

$$\hat{\beta} \sim \mathcal{N}(\beta, \sigma^2(\tilde{X}^T\tilde{X})^{-1}), \tag{5.25}$$

ou de manière équivalente

$$\hat{\beta} \sim \mathcal{N}(\beta, \sigma^2(X^T\Sigma X)^{-1}). \tag{5.26}$$

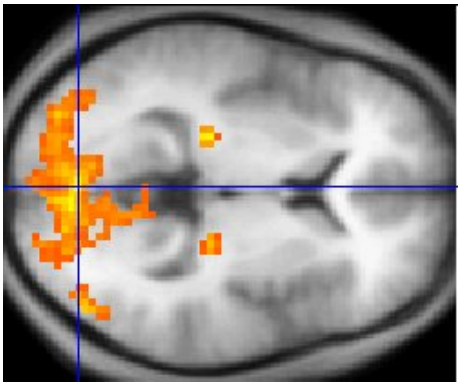


FIGURE 5.7 – Carte d'activation pour une stimulation visuelle. Source : http ://en.wikipedia.org/wiki/Functional_magnetic_resonance_imaging.

5.3.3 Inférence

Le modèle défini dans la section précédente permet de mener des *tests d'hypothèses* sur les paramètres β. On peut considérer par exemple le test relatif à l'activation du stimulus de type 1 :

$$\begin{cases} H_0 & : & \beta_1 = 0 \\ H_1 & : & \beta_1 > 0 \end{cases} \tag{5.27}$$

L'hypothèse nulle H_0 correspond à l'affirmation que le stimulus de type 1 n'a pas d'effet sur le voxel considéré, alors que l'hypothèse alternative H_1 correspond à sa négation. Ce type de test permet notamment d'obtenir des *cartes d'activation* pour l'ensemble du cerveau, et déterminer quelles sont les zones du cerveau qui sont activées suite à un certain type de stimulus. La représentation d'une telle carte d'activation est faite sur la figure 5.7.

De manière plus générale, on peut mener des tests d'hypothèses sur n'importe quelle combinaison linéaire, ou *contraste*, des paramètres. On définit généralement le contraste d'intérêt grâce à un vecteur c de longueur Q, et on peut définir le test d'hypothèse relatif à c par

$$\begin{cases} H_0 & : & c^T\beta = 0 \\ H_1 & : & c^T\beta > 0 \end{cases}, \tag{5.28}$$

pour un test unilatéral et par

$$\begin{cases} H_0 & : & c^T\beta = 0 \\ H_1 & : & c^T\beta \neq 0 \end{cases} \tag{5.29}$$

pour un test bilatéral.

Le premier exemple peut être effectué grâce au contraste $c = (1\ 0\ \cdots\ 0)^T$, car $c^T\beta = \beta_1$ et on retombe sur le cas du test (5.28). Mais les tests par contraste permettent aussi par exemple de tester si deux types de stimuli, par exemple β_1 et β_2 ont un effet identique sur l'activité cérébrale d'un voxel donné ou non. Ce test peut être effectué en définissant le contraste $c = (1\ -1\ 0\ \cdots\ 0)^T$, car alors $c^T\beta = \beta_1 - \beta_2$ et l'hypothèse nulle est $H_0 : \beta_1 = \beta_2$ et l'hypothèse alternative pour un test bilatéral est $H_1 : \beta_1 \neq \beta_2$. Ce contraste particulier est appelé le contraste *différentiel*. Un dernier exemple est celui du contraste $c = (1\ 1\ 0\ \cdots\ 0)^T$, qui permet de tester si les stimuli de type 1 et 2 pris ensemble ont un effet sur le voxel considéré.

Pour tester l'hypothèse (5.28) ou (5.29), il faut définir une *statistique* qui permette de prendre une décision suivant le seuil de tolérance fixé. Pour ce faire, il est nécessaire de connaître la distribution de $c^T\hat{\beta}$ sous l'hypothèse nulle. Supposons être dans le cadre de données non-corrélées, c'est-à-dire telles que $\Sigma = I$. On peut montrer facilement que $c^T\hat{\beta}$ suit une distribution normale de moyenne $c\beta$ et de variance $\sigma^2 c^T(X^TX)^{-1}c$ sous l'hypothèse nulle. Cependant, en pratique la valeur de la variance σ^2 n'est pas connue a priori, et il n'est pas possible d'utiliser la statistique normale pour le test. Il faut donc trouver un estimateur $\hat{\sigma^2}$ permettant de déduire la valeur de σ^2 à partir des données. Une approche possible est de définir celui-ci à partir des *résidus* $\rho := Y - X\hat{\beta}$ de la minimisation des moindres carrés par

$$\hat{\sigma}^2 = \frac{\rho^T\rho}{n-Q}. \tag{5.30}$$

Dès lors, on peut définir la statistique t par

$$t = \frac{c^T\hat{\beta}}{\sqrt{\hat{\sigma}^2 c^T(X^TX)^{-1}c}}, \tag{5.31}$$

qui suit une distribution de student à $n - (Q+1)$ degrés de liberté sous l'hypothèse nulle. Il est alors possible de calculer les *p-valeurs* pour les tests unilatéral et bilatéral respectivement par $p := P(t_{n-(Q+1)} > t_{obs})$ et $p := P(t_{n-(Q+1)} > |t_{obs}|)$, où $t_{n-(Q+1)}$ désigne une variable aléatoire de student à $n-(Q+1)$ degrés de liberté. Étant donné un seuil de tolérance α, la règle de décision relative au rejet de l'hypothèse nulle sera donnée par $p < \alpha$.

5.4 Optimisation

L'estimation des paramètres $\beta_1, \ldots, \beta_Q$ donnée par l'équation (5.14) dépend des données particulières traitées, et varie donc d'un jeu de données à l'autre. Le but de l'optimisation annoncée dans ce chapitre est d'avoir des estimations les plus correctes possible, c'est-à-dire les plus proches de leurs vraies valeurs,

de manière à ce que les tests d'hypothèse, qui dépendent de ces valeurs, soient les plus *fiables* possible. Comme nous allons le voir dans les sections suivantes, la "fiabilité" des estimations dépend du schéma d'expérience. Le but sera donc de trouver les schémas d'expérience optimaux pour la "fiabilité" des estimations. Par exemple, pour le problème des cartes d'activation, cela revient à vouloir minimiser le nombre de *faux-positifs* obtenus.

5.4.1 Critères d'optimalité

La mesure principale de l'écart des estimations à la vraie valeur β peut être faite grâce à la matrice de variance-covariance de l'estimateur $\hat{\beta}$. Pour des données indépendantes, elle est donnée par

$$Cov(\hat{\beta}) = \sigma^2 (X^T X)^{-1}. \tag{5.32}$$

Lorsque les données sont temporellement corrélées, on applique un filtre temporel comme décrit dans la section précédente et on obtient une matrice de variance-covariance

$$Cov(\hat{\beta}) = \sigma^2 (X^T \Sigma X)^{-1}. \tag{5.33}$$

Si on s'intéresse à un contraste d'intérêt c, il faut considérer la variance de $c^T\hat{\beta}$, c'est-à-dire

$$var(c^T\hat{\beta}) = \sigma^2 c^T (X^T \Sigma X)^{-1} c. \tag{5.34}$$

Plus celle-ci sera petite, plus les estimations de $c^T\beta$ seront proches de leur vraie valeur, et plus le test sera fiable. On définit donc la mesure d'*efficacité*, que l'on note F_1, comme l'inverse de la variance de $c^T\hat{\beta}$ pour un contraste donné :

$$F_1(c, X) = (c^T (X^T \Sigma X)^{-1} c)^{-1}, \tag{5.35}$$

où l'on a omis la quantité σ^2 car celle-ci est supposée constante et n'interviendra donc pas dans l'optimisation.

Dans cette définition, on voit clairement la dépendance de l'efficacité de l'estimateur avec la matrice X, et donc directement, par construction de cette matrice, avec l'expérience $s(.)$. On va donc tenter de trouver des expériences, dans l'ensemble des expériences par événements vérifiant ISI=TR, qui maximisent l'efficacité.

5.4.2 Contraintes

En pratique, la recherche d'expériences optimales est restreinte par un certain nombre de *contraintes psychologiques*. Par exemple, il est préférable d'éviter

un nombre trop grand de répétition d'un même stimulus à la suite, pour éviter l'accoutumance du sujet au stimulus, qui apporte un comportement non-linéaire du signal qui n'est pas pris en compte dans le modèle. On veut également éviter que l'expérience soit prévisible par le sujet, car de nouveau le modèle utilisé pourrait sortir de son domaine d'application. Enfin, on peut vouloir imposer a priori les fréquences d'apparition pour chaque type de stimulus. Pour s'assurer que ces contraintes soient vérifiées, on définit des quantités qui permettent de mesurer avec quel degré une expérience donnée respecte celles-ci. On considère ici deux contraintes : le *contrebalancement* et la *fréquence*.

Contrebalancement

Le contrebalancement est une mesure de la non-prédictibilité d'une expérience. Dans une expérience avec un bon contrebalancement, la probabilité d'apparition d'un certain type de stimulus à chaque moment de l'expérience doit être indépendante de ce qui s'est déjà produit auparavant (sans considérer les périodes de repos).

Pour un contrebalancement dit du *premier ordre*, cela peut se traduire par la condition que les stimuli de type j doivent succéder aux stimuli de type i avec une probabilité p_{ij} qui vérifie

$$p_{ij} = p_i p_j, \tag{5.36}$$

où p_i et p_j sont les probabilités d'apparition des stimuli de type i et j respectivement, pour $i, j = 1, \ldots, Q$. Ces probabilités sont fixées par les fréquences d'apparition désirées pour chaque type de stimulus $f_1, \ldots, f_Q$. Par exemple, pour une expérience où tous les stimuli ont la même fréquence d'apparition, on aura $p_{ij} = 1/Q^2$ et tous les types de stimuli doivent se suivre avec la même fréquence.

L'espérance du nombre n_{ij} de stimuli de type j devant succéder à des stimuli de type i peut alors être donnée par

$$E(n_{ij}) = np_{ij} = np_i p_j = nf_i f_j, \tag{5.37}$$

où n désigne le nombre total de stimuli présentés sur l'ensemble de la session.

Il suffit alors de compter le nombre n_{ij} de stimuli de type j succédant à des stimuli de type i pour l'expérience considérée pour pouvoir calculer l'écart à sa valeur idéale $(n_{ij} - E(n_{ij}))$. En faisant de même pour chaque type de stimulus, on peut définir une notion de distance du premier ordre de l'expérience considérée à un contrebalancement parfait par

$$d_1(s(.)) := \sum_{i,j=1}^{Q} (n_{ij} - E(n_{ij}))^2. \tag{5.38}$$

On peut généraliser cette notion aux ordres supérieurs en considérant le nombre de stimuli de type j succédant un type i, mais avec un décalage de r stimuli, r étant alors l'*ordre* du contrebalancement considéré. De manière analogue à l'ordre 1, on peut calculer l'espérance du nombre de stimuli de type j succédant à des stimuli de type i avec décalage r, que l'on note $E(n_{ijr})$, ainsi que sa valeur observée n_{ijr}, et alors définir

$$d_r(s(.)) := \sum_{i,j=1}^{Q} (n_{ijr} - E(n_{ijr}))^2. \tag{5.39}$$

Le contrebalancement d'ordre maximum R peut alors être défini par (cf. [Wager and Nichols, 2003], pp. 302)

$$\begin{aligned} F_2(s(.)) &:= 1 - \frac{1}{Q^2 R} \sum_{r=1}^{R} d_r(s(.)) \\ &= 1 - \frac{1}{Q^2 R} \sum_{r=1}^{R} \sum_{i,j=1}^{Q} (n_{ijr} - E(n_{ijr}))^2. \end{aligned} \tag{5.40}$$

Le fait de prendre la différence avec 1 dans la définition permet d'avoir une quantité à *maximiser*.

Fréquences

On peut aussi vouloir imposer certaines fréquences pour chaque type de stimulus. Pour quantifier la manière dont une expérience respecte les fréquences imposées, on peut définir la quantité (cf. [Wager and Nichols, 2003], pp. 302)

$$F_3(s(.)) := 1 - \sum_{i=1}^{Q} |n_i - n f_i|, \tag{5.41}$$

où n_i désigne le nombre de stimuli de type i observés, n le nombre total de stimuli et f_i les fréquences imposées pour chaque type de stimulus (de nouveau sans considérer les repos).

5.4.3 Robustesse

En plus d'avoir des expériences efficaces (au sens de la définition donnée par l'équation (5.35)), on peut vouloir des expériences qui soient également *robustes* par rapport à la définition de la fonction HRF.

En effet, la réponse hémodynamique varie d'un individu à l'autre, et d'un voxel à l'autre (par exemple [Aguirre et al., 1998] ou [Miezin et al., 2000]). Cela a pour effet que la réponse calculée par l'équation (5.5), et qui dépend directement de la fonction HRF canonique h_c choisie, contient une erreur lorsque la réponse HRF réelle h_t est différente de h_c. Cela aura pour conséquence que le modèle GLM sera biaisé et qu'il sera impossible de déterminer

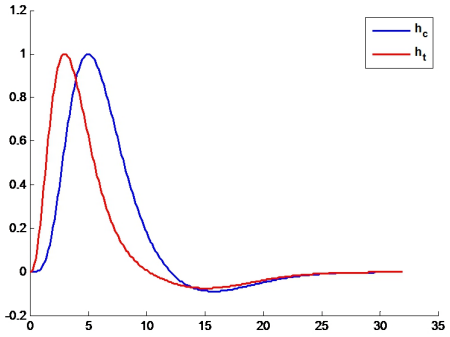


FIGURE 5.8 – Représentation de la fonction HRF canonique h_c (en bleu) et de la fonction HRF réelle h_t (en rouge).

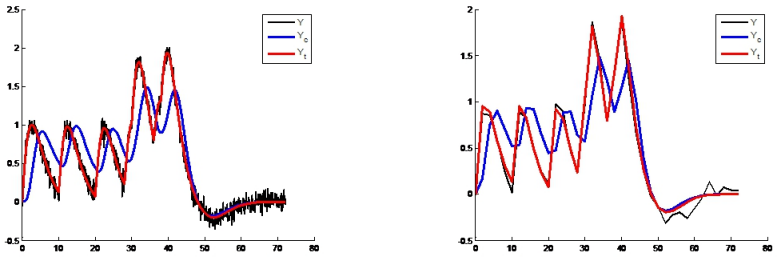


FIGURE 5.9 – Illustration de l'erreur commise par l'utilisation d'une fonction HRF erronée pour le modèle GLM sur (1) les données continues et (2) les données discrétisées aux temps TR.

un paramètre β tel que l'erreur résiduelle soit de moyenne nulle. Pour mettre ceci en avant, considérons des données Y, représentée en noir sur la figure 5.9 (données continues sur l'image de gauche, et discrétisées au temps d'acquisition sur l'image de droite). Supposons que la fonction HRF réelle h_t soit différente de la fonction HRF canonique h_c, et en particulier dans ce cas que la durée au sommet TP réelle soit plus petite pour h_t que pour h_c, comme représenté sur la figure 5.8. Les prédictions faites par le GLM avec chacune des fonctions HRF sont faites sur la figure 5.9, en rouge avec la fonction h_t et en bleu avec h_c. On voit clairement sur cette figure que la prédiction $X_c\beta_c$ faite grâce à h_c ne colle pas du tout aux données, contrairement à la prédiction $X_t\beta_t$ faite grâce à h_t. Notons que cet exemple est fictif car il n'est pas possible en pratique d'avoir les données continues illustrées sur l'image de gauche, mais il permet d'illustrer l'erreur due à l'utilisation d'une fonction HRF erronée.

Tentons de donner une mesure de l'erreur commise par l'utilisation d'une fonction HRF erronée. Supposons pour cela que la fonction HRF canonique h_c ne permette pas de prédire les données, *i.e.* soit telle que

$$\nexists \beta_c \in \mathbb{R}^Q \text{ tel que } E(Y) = X_c\beta_c, \tag{5.42}$$

où l'on note X_c la matrice des prédictions calculée à partir de la fonction HRF h_c.

Supposons que la fonction HRF réelle soit donnée par la fonction h_t. La relation (5.42) implique que

$$E(Y) = X_t\beta_t \neq X_c\beta_c, \ \forall \beta_c \in \mathbb{R}^Q. \tag{5.43}$$

Une mesure de l'erreur commise par l'utilisation de h_c dans le GLM est donnée par

$$\begin{aligned}
\mu(X_c, X_t, \beta_t) &:= \|E(Y - X_c\hat{\beta}_c)\| \\
&= \|E(Y) - E(X_c\hat{\beta}_c)\| \\
&= \|X_t\beta_t - X_cE(\hat{\beta}_c)\| \\
&= \|X_t\beta_t - X_cE((X_c^TX_c)^{-1}X_cY)\| \\
&= \|X_t\beta_t - X_c(X_c^TX_c)^{-1}X_cE(Y)\| \\
&= \|X_t\beta_t - X_c(X_c^TX_c)^{-1}X_cX_t\beta_t\| \\
&= \|(I - \mathcal{P}(X_c))X_t\beta_t\|,
\end{aligned} \tag{5.44}$$

où $\mathcal{P}(X_c)$ désigne l'opérateur de projection orthogonale sur le sous-espace vectoriel engendré par les colonnes de la matrice X_c.

Cependant, dans le contexte de ce travail, il est important de développer des outils qui ne font pas intervenir les données réelles Y. En effet, dans le processus d'optimisation, nous considérerons un grand nombre de candidats possibles, et donc beaucoup d'expériences $s(.)$ différentes, si bien qu'il est impossible d'avoir les données Y pour chacune des expériences rencontrées (puisque chaque jeu de données implique une session d'une vingtaine de minutes, et le prix de fonctionnement de la machine). Pour la quantité $\mu(X_c, \beta_t)$, il est possible de contourner la dépendance avec β_t (et donc les données réelles Y puisque $\hat{\beta}_t = (X_t^TX_t)^{-1}X_tY$) en considérant le *pire des cas*. Cela se traduit par le fait de prendre le maximum parmi l'ensemble des vecteurs β_t dans la définition de $\mu(X_c, \beta_t)$:

$$\mu(X_c, X_t) := \max_{\beta_t} \|(I - \mathcal{P}(X_c))X_t\beta_t\| \overset{\text{déf}}{=} \|(I - \mathcal{P}(X_c))X_t\|, \tag{5.45}$$

où la dernière norme est la norme *matricielle* de la matrice $(I - \mathcal{P}(X_c))X_t$.

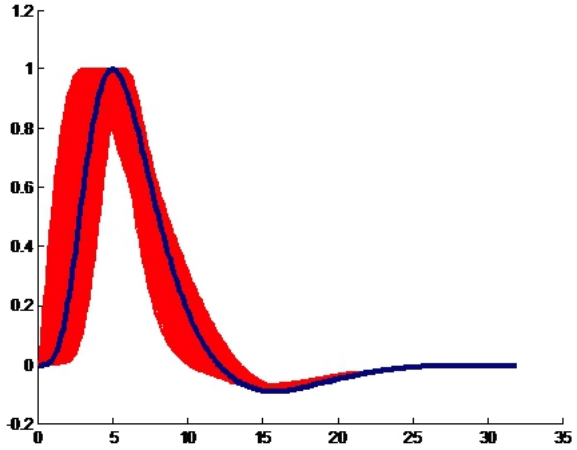


FIGURE 5.10 – Illustration du voisinage (zone rouge) de la fonction HRF canonique (en bleu) considéré.

Enfin, en pratique on ne connaît pas non plus la vraie fonction HRF h_t (sinon on l'utiliserait directement au lieu de h_c). Une solution est de tenter de trouver une expérience qui minimise $\mu(X_c)$ pour l'ensemble des fonctions h_t définies dans un *voisinage* de la fonction HRF canonique h_c. Une telle expérience sera *robuste* par rapport à la définition de la fonction HRF puisque l'erreur commise par l'utilisation de h_c sera minimale pour n'importe quelle fonction HRF h_t définie dans le voisinage considéré.

Le voisinage utilisé a été défini grâce aux données provenant des articles [Aguirre et al., 1998] et [Miezin et al., 2000], qui présentent des variations de fonctions HRF pour des individus différents, et pour des régions du cerveau différentes. Nous avons alors déterminé les paramètres θ de la double fonction gamma (5.19) de telle manière à couvrir l'ensemble des fonctions HRF présentées dans ces articles. Notons Θ l'ensemble des valeurs des paramètres θ qui définit ce voisinage. La représentation du voisinage de la fonction HRF canonique est faite sur la figure 5.10, où la courbe bleue représente la fonction HRF et la zone rouge représente le voisinage considéré.

La mesure de robustesse pour une expérience $s(.)$ donnée peut donc être définie par

$$F_4(s(.)) := (\mu(s(.)))^{-1} = \left(\underset{\theta \in \Theta}{\text{moyenne}}(\mu(X_c, X_t(\theta))) \right)^{-1}, \qquad (5.46)$$

où l'on a pris l'inverse de la quantité μ pour avoir une quantité à *maximiser*.

5.5 Algorithme génétique

Passons maintenant à l'aspect algorithmique pour la résolution du problème d'optimisation défini dans la section précédente. Cette section décrit la structure de l'algorithme génétique utilisé, avec dans la section 5.5.1 la question du codage des individus, dans la section 5.5.2 la définition de la fonction objectif utilisée, et dans la section 5.5.3 les détails d'implémentation pertinents.

5.5.1 Codage des individus

Considérons tout d'abord la question de la représentation des individus. L'ensemble des solutions admissibles, que l'on note Ξ dans cette section pour ne pas confondre avec la matrice X du GLM, est défini comme l'ensemble des expériences $s(.)$ de durée $t = n \cdot TR$ secondes, où n désigne le nombre de données obtenues sur la session, telles que l'ISI est constant et égal à TR. On suppose également qu'il n'y a au plus qu'un stimulus à chaque temps $t = i \cdot \mathrm{TR}$ (il pourrait éventuellement y avoir des stimuli de types différents définis au même temps avec la définition (5.3)). Il est donc possible de coder les expériences dans un vecteur b de taille n, où la i^e composante contient le type de stimulus soumis au temps $i \cdot \mathrm{TR}$, c'est-à-dire le vecteur défini par

$$\begin{cases} [b]_i = j & \text{si } \exists j \in \{1, \dots, Q\} : \ s_j(i \cdot \mathrm{TR}) = \delta, \\ [b]_i = 0 & \text{sinon.} \end{cases} \tag{5.47}$$

L'ensemble des strings S dans la définition de la section 2.2.1 vaut donc

$$S = \{0, \dots, Q\}^n. \tag{5.48}$$

5.5.2 Fonction de fitness

Définissons maintenant la fonction de fitness utilisée pour l'algorithme génétique. Dans la section 5.4 sont développés plusieurs critères pour traduire la qualité que l'on espère retrouver dans les expériences. Optimiser en fonction de tous ces critères simultanément relève d'une optimisation *multi-objectifs*.

Plusieurs méthodes peuvent être utilisées pour réaliser une telle optimisation. Celle que nous avons choisie est la méthode qui consiste à réaliser la recherche sur une *combinaison convexe* fixée des quatre critères individuels considérés. Pour permettre une cohérence entre les ordres de grandeur de chaque critère, ceux-ci sont d'abord *standardisés* par l'équation (2.4), *i.e.*

$$\tilde{F}_i(X) := \frac{F_i(X) - \min\limits_{s \in \Xi} F_i(X)}{\max\limits_{s \in \Xi} F_i(X) - \min\limits_{s \in \Xi} F_i(X)}, \ i = 1, \dots, 4, \tag{5.49}$$

où X est la matrice du GLM d'une expérience $s(.)$ donnée.

Cette définition présuppose la connaissance des valeurs minimale et maximale pour chacun des critères. Celles-ci peuvent être obtenues grâce à une optimisation mono-objectif de l'algorithme génétique en considérant séparément chaque F_i comme fonction objectif.

Une fois ces optimisations préliminaires effectuées, et étant donné un vecteur de poids $w = (w_1, w_2, w_3, w_4)$ tel que $w_i \geq 0$ et $\sum_i w_i = 1$, on peut définir la fonction de fitness par

$$F(X) := \sum_{i=1}^{4} w_i \tilde{F}_i(X). \tag{5.50}$$

L'optimisation multi-objectifs par combinaison convexe permet en fait de trouver les *sommets* de l'enveloppe convexe formée par l'ensemble des points $\{(\tilde{F}_1(X), \tilde{F}_2(X), \tilde{F}_3(X), \tilde{F}_4(X)) : \quad s \in \Xi\}$, et donc de trouver les meilleures expériences pour l'ensemble des critères considérés. Le point particulier de l'enveloppe convexe qui sera atteint dépend de la définition des poids $w_i \geq 0$ qui désignent l'importance relative que l'on porte à chaque critère. Notons que souvent les objectifs considérés sont *contradictoires*, c'est-à-dire que l'amélioration d'un critère fait décroître les autres critères, et les solutions trouvées constitueront en fait des *compromis* entre les différents objectifs.

5.5.3 Structure algorithmique

L'algorithme utilisé est celui qui avait été implémenté dans le cadre de l'article de [Wager and Nichols, 2003], et qui est disponible à l'adresse suivante : http ://psych.colorado.edu/∼tor/Software/genetic_algorithms.html. Présentons d'abord le fonctionnement général de celui-ci. Nous donnerons ensuite les améliorations qui y ont été apportées.

Algorithme génétique de Wager & Nichols (GAWN)

L'algorithme génétique de Wager & Nichols consiste en un ensemble de fichiers qui forment la toolbox *OptimizeDesign*. Ces fichiers consistent d'une part en l'implémentation du GA en lui-même ainsi que des fonctions de sélection, de reproduction, de mutation et de fitness, ainsi qu'un ensemble de fonctions utilisées pour tous les calculs intermédiaires, par exemple le calcul de la matrice X intervant dans le GLM et nécessaire pour le calcul de la fonction de fitness. Ces fonctions font parfois appel à des fonctions issues du célèbre programme SPM (Statistical Parametric Mapping), qui est un package MATLAB très complet pour l'imagerie médicale. Il est donc assez difficile de se "plonger" dans cet ensemble de fonctions, c'est pourquoi nous ferons une présentation simplifiée, en se concentrant sur le point de vue de l'algorithme génétique. Plus de détails peuvent être trouvés dans le fichier *readme.doc* de

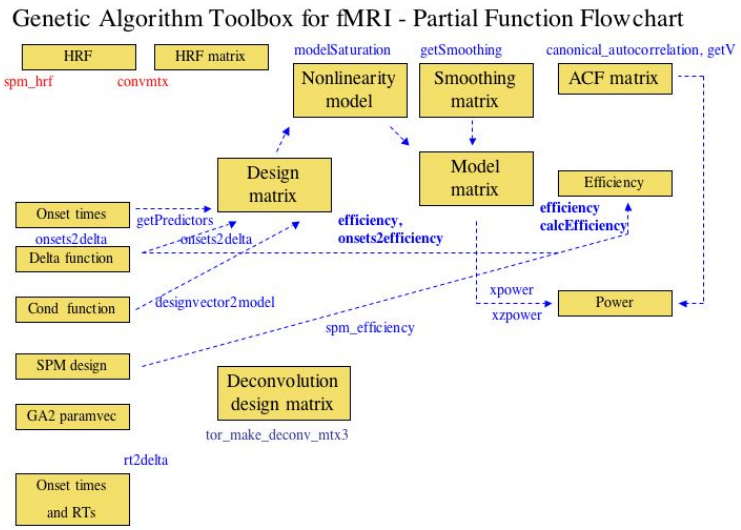


FIGURE 5.11 – Organigramme (partiel) de la toolbox *OptimizeDesign*. Source : http ://psych.colorado.edu/∼tor/Software/genetic_algorithms.html.

la toolbox *OptimizeDesign*, dans l'article [Wager and Nichols, 2003], ou évidemment dans le code lui-même. L'organigramme partiel de l'*OptimizeDesign* est donné sur la figure 5.11 pour donner un aperçu des relations entre les différents groupes de fonctions.

Avant de pouvoir donner la structure algorithmique utilisée, donnons la stratégie de sélection proposée par [Wager and Nichols, 2003]. Soit $B_t = (b_{1,t}, \ldots, b_{m,t})$ une population d'individus à un temps t et F une fonction de fitness. On définit $F^\star$ comme la transformation de la fitness par une sigmoïde, plus un terme aléatoire uniforme dans [0, 1] :

$$F^\star := \mathrm{rand}(0,1) + \frac{1}{1+\exp(-\alpha F_s)}, \tag{5.51}$$

où α est une constante représentant la *pression de sélection* et où F_s désigne la fitness *standardisée* de F sur l'ensemble B_t, c'est-à-dire la fonction définie par

$$F_s(b_{i,t}) := \frac{F(b_{i,t}) - \mu(F(B_t))}{\sigma(F(B_t))}, \tag{5.52}$$

où $\mu(F(B_t))$ est la moyenne de la fitness F sur les éléments de B_t, *i.e.*

$$\mu(F(B_t)) = \frac{1}{m}\sum_{j=1}^{m} F(b_{j,t}), \tag{5.53}$$

et où $\sigma(F(B_t))$ est la déviation de F sur les éléments de B_t, *i.e.*

$$\sigma(F(B_t)) = \left(\frac{1}{m} \sum_{j=1}^{m} (F(b_{j,t}) - \mu)^2 \right)^{\frac{1}{2}}. \tag{5.54}$$

La stratégie de sélection SWN consiste à choisir deux fois les $\frac{m}{2}$ individus dont la fitness *transformée* $F^\star$ est au-dessus de la médiane pour former les m individus de B_S. L'utilisation du terme aléatoire rand$(0,1)$ dans la fonction transformée $F^\star$ permet d'inclure une certaine variation dans la sélection des individus. Il est affirmé dans [Wager and Nichols, 2003] que le paramètre α permet quant à lui de déterminer le poids relatif de la contribution aléatoire dans la fitness transformée. Plus précisément il est soutenu que lorsque $\alpha = 0$, le terme de droite de (5.51) est constant et la sélection est équivalente à un tirage aléatoire dans les éléments de B_t, alors que lorsque $\alpha \to +\infty$, la sélection de la médiane supérieure de $F^\star$ est équivalente à la sélection de la médiane de F. Cependant, cette dernière affirmation n'est pas vraie tout le temps. Elle n'est vrai que lorsque la moyenne $\mu(B_t)$ de F sur B_t est assez proche de la médiane que pour diviser les éléments de B_t en deux groupes de même nombre d'éléments. En effet, lorsque α est très grand dans l'équation (5.51), les éléments de fitness standardisée $F_s < 0$ auront une fitness transformée

$$\begin{aligned} F^\star &= \text{rand}(0,1) + (1 + exp(-\alpha F_s))^{-1} \\ &\simeq \text{rand}(0,1) + (1+\infty)^{-1} \\ &= \text{rand}(0,1), \end{aligned} \tag{5.55}$$

alors que les éléments de fitness standardisée $F_s > 0$ auront une fitness transformée

$$\begin{aligned} F^\star &= \text{rand}(0,1) + (1 + exp(-\alpha F_s))^{-1} \\ &\simeq \text{rand}(0,1) + (1+0)^{-1} \\ &= \text{rand}(0,1) + 1. \end{aligned} \tag{5.56}$$

On peut dès lors affirmer que si la fonction F_s divise les individus de B_t en deux groupes de même nombre, caractérisés respectivement par des valeurs positives et négatives de F_s, alors la sélection SWN est équivalente à la sélection de la médiane supérieure de F. Cependant, la moyenne d'un ensemble de valeurs n'est pas toujours égale (ou assez proche) de sa médiane, et donc l'affirmation précédente n'est pas vraie. De plus, si une grande majorité d'éléments ont une fonction F_s de même signe, la sélection par SWN avec des grandes valeurs de α est presque équivalente à un tirage aléatoire.

Pour confirmer ces affirmations, nous avons tracé sur la figure 5.12 le pourcentage d'individus identiques entre la médiane supérieure de $F^\star$ et la

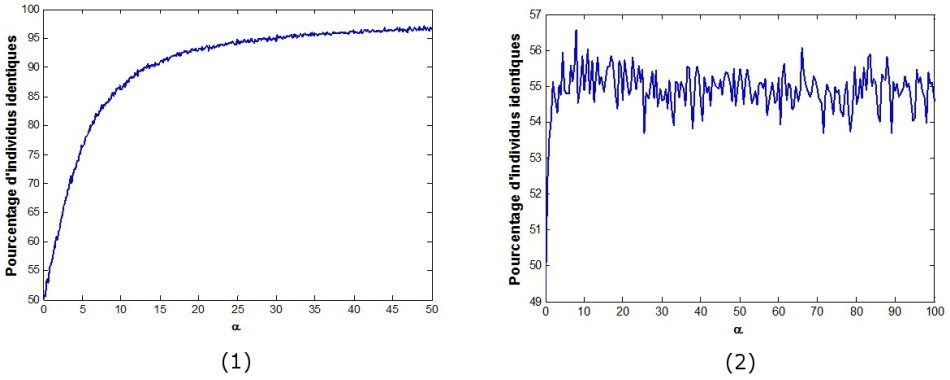


FIGURE 5.12 – Représentation du pourcentage d'individus identiques entre la sélection de la médiane supérieure de $F^\star$ et la médiane supérieure de F, en fonction du paramètre α, (1) pour un ensemble d'individus identiquement distribués autour de 0 et (2) pour un ensemble d'individus distribués à 90% d'individus où $F_s > 0$ et 10% où $F_s < 0$.

médiane de F, en fonction de la valeur du paramètre α. Nous avons considéré pour la courbe 5.12(1) un ensemble de 100 individus tel que $F_s > 0$ pour une moitié et $F_s < 0$ pour l'autre moitié, et pour la courbe 5.12(2) un ensemble de 100 individus tel que F_s est positif pour 90 individus et négatif pour les 10 individus restants. On peut voir que la première courbe est croissante en fonction de la valeur de α, comme annoncé par [Wager and Nichols, 2003], mais la deuxième courbe oscille autour de 55% environ. Il est donc clair que l'hypothèse de la "bonne répartition" de F_s autour de 0 est primordiale pour le bon fonctionnement de la méthode.

Cependant, cette faiblesse dans la stratégie de sélection proposée ne pose pas de problème pour l'algorithme, et ne fausse pas les résultats de l'article, car il n'y a que si quelques valeurs sont fort éloignées des autres que cela décale assez la moyenne que pour avoir une sélection plus aléatoire. Pour le confirmer nous avons calculé le pourcentage d'individus tels que $F_s > 0$ à chaque étape de sélection de l'algorithme pour une exécution typique de l'algorithme. On peut voir sur la figure 5.13 que celui-ci varie grossièrement entre 40 et 50%, ce qui ne change que peu le comportement de la stratégie de sélection puisque la valeur idéale est de 50%. La stratégie de sélection SWN est donc valide, mais il serait sans doute plus correct d'utiliser une stratégie moins "fragile", comme par exemple une stratégie par roulette ou par tournoi, ou, pour rester dans le même état d'esprit que la stratégie SWN, choisir la médiane supérieure de la fonction transformée

$$F^\star := \alpha \text{rand}(0,1) + (1-\alpha)F_n, \tag{5.57}$$

où F_n désigne la fitness F normalisée dans $[0,\ 1]$, et où $\alpha \in [0,\ 1]$. Avec cette

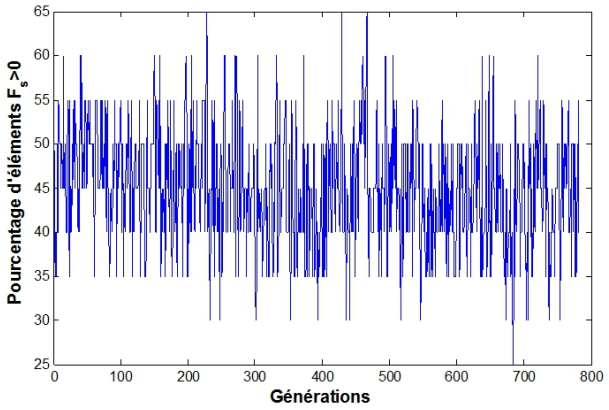


FIGURE 5.13 – Représentation du pourcentage d'individus tels que $F_s > 0$ dans la population B_t, en fonction des générations t.

définition, il est clair que la valeur de α détermine la contribution aléatoire dans le processus de sélection.

Nous avons cependant utilisé la stratégie SWN dans ce travail car nous nous sommes rendu compte de sa faiblesse seulement après avoir obtenu tous les résultats, mais il serait donc intéressant dans des travaux ultérieurs d'utiliser une stratégie plus "solide". Précisons enfin que la valeur α utilisée a été fixée à $\alpha = 2.1$ puisque c'est la valeur qui semble donner les meilleurs résultats selon [Wager and Nichols, 2003].

On définit aussi la stratégie de mutation par N_M-*permutation* comme l'opérateur qui consiste à permuter N_M gènes choisis de manière aléatoire dans l'individu considéré. Une illustration d'une telle mutation pour $N_M = 3$ est faite sur la figure 5.14.

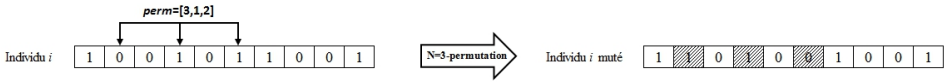


FIGURE 5.14 – Représentation d'une mutation par 3-permutation.

Donnons maintenant la structure d'algorithme utilisée. Pour cela on se fixe une taille de problème n, qui correspond au nombre de scans réalisés sur une session de longueur $t = n \cdot \mathrm{TR}$ secondes. On se fixe également un nombre Q de types différents de stimuli, ainsi que des fréquences $f_1, \ldots, f_Q$ d'apparition de ces stimuli (si $\sum f_i < 1$, certains temps seront vides, et correspondront à des repos). L'algorithme GAWN est donné par la spécification 5.1.

Algorithme 5.1:
Soient donnés $m, t_{\max} \in \mathbb{N}_0$, avec m pair.

1. **Initialisation :** Créer une population initiale $B_0 = (b_{1,0}, \ldots, b_{m,0})$ de taille m, avec des individus choisis aléatoirement dans S ;
2. **Corps de l'algorithme :** Pour $t = 0$ jusque t_{max} :
   - 2.1. **Calcul de la fitness :** Calculer la fitness F définie par l'équation (5.50) pour chaque individu de la population ;
   - 2.2. **Sélection :** sélectionner m individus $(b_{1,t+1}, \ldots, b_{m,t+1})$ dans B_t par la stratégie de sélection SWN avec pression $\alpha = 2.1$;
   - 2.3. **Reproduction :** pour $i = 1$ jusque $m - 1$ par pas de 2, remplacer les parents $b_{i,t+1}$ et $b_{i+1,t+1}$ par leurs 2 enfants issus d'un crossover à 1 point de coupure ;
   - 2.4. **Mutation :** pour $i = 1$ jusque $m - 1$, muter $b_{i,t+1}$ par une 2-permutation avec une probabilité de $p_M = 0.01$.
   - 2.5. **Élitisme :** remplacer $b_{l,t+1}$ par $b_{l,t} = \text{argmax}_{b \in B_t} F(b)$ si $F(b_{l,t+1}) < F(b_{l,t})$;
   - 2.6. **Vérification de la dispersion de la population :** Calculer le coefficient de corrélation ρ entre les individus. Si $\rho < 0.4$, réaliser une mutation de 10% de la population par 1-inversion.

Améliorations apportées

Donnons maintenant les améliorations apportées à cet algorithme. La première modification que nous avons apportée est dans la définition de la fitness. En effet, à l'origine, la standardisation de chaque critère de la fitness dans GAWN est faite grâce à un sigma-scaling (cf. section 2.4.1), et dépend donc du reste des individus dans la population. La recherche multi-objectif effectuée avec cette définition n'est pas très stable car la standardisation change à chaque génération, ce qui rend impossible une comparaison consistante. Notre définition permet d'éviter cet inconvénient. Cela permet notamment d'avoir un GA dont la fitness du meilleur individu est croissante au cours des générations, ce qui n'est pas le cas avec l'autre définition.

Ensuite, nous avons considéré un autre opérateur de reproduction que le crossover à un point. En effet, l'idée est de définir un opérateur qui permet de générer des enfants qui respectent les fréquences données. Cela s'avère être beaucoup plus efficace que la stratégie qui consiste à inclure un terme de pénalisation dans la fonction de fitness. L'opérateur créé s'inspire du fonctionnement de l'opérateur *order-based crossover operator OX1* pour le TSP ([Larrafiaga et al., 1996]). L'idée est de construire un enfant en choisissant une sous-partie du premier parent, supprimer chaque élément de cette sous-

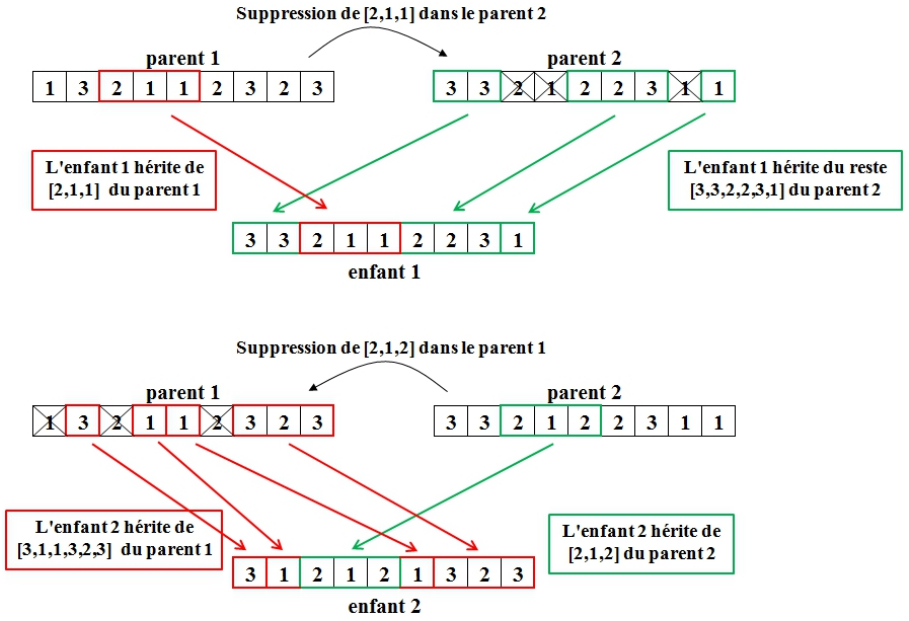


FIGURE 5.15 – Illustration du crossover inspiré du OX1.

partie dans le deuxième parent, et enfin compléter l'enfant par les éléments restants du deuxième parent dans l'ordre dans lesquels ils apparaissent. Le même peut être fait pour un deuxième enfant en inversant les rôles des parents.

Par exemple, considérons les parents $p_1 = (1, 3, 2, 1, 1, 2, 3, 2, 3)$ et $p_2 = (3, 3, 2, 1, 2, 2, 3, 1, 1)$ dont chaque gène possède un allèle dans $\{1, 2, 3\}$, et dont chaque allèle est représenté trois fois. Considérons la sous-partie de p_1 composée des gènes allant de la position 3 à 5 : $e_1 := (*, *, 2, 1, 1, *, *, *, *)$. On supprime les éléments $2, 1, 1$ dans le deuxième parent dans l'ordre d'apparition de chiffres. On obtient $(3, 3, *, *, 2, 2, 3, *, 1)$. On complète maintenant l'enfant e_1 par les éléments de p_2 dont les éléments de e_1 ont été supprimés pour obtenir : $(3, 3, 2, 1, 1, 2, 2, 3, 1)$. On peut faire de même en inversant les rôles des deux parents pour obtenir $e_2 = (3, 1, 2, 1, 2, 1, 3, 2, 3)$. Une illustration de cette reproduction est faite sur la figure 5.15.

Il est clair que le crossover ainsi défini préserve les fréquences des parents. Il n'est donc plus nécessaire de considérer le terme F_3 relatif à la fréquence dans la définition de la fitness. Pour mettre en avant l'amélioration au niveau des performances du GA due à cette nouvelle définition du crossover, on considère un problème simple pour lequel on demande la préservation des fréquences. Le problème est celui de trouver le vecteur binaire contenant la plus longue chaîne de "1", où le nombre de "0" et de "1" doit être identique

dans le vecteur. On définit la fonction de fitness comme

$$f(v) := \sum_{\overline{v} \subset_1 v} (\sum_{i=1}^{l(\overline{v})} (\overline{v})), \tag{5.58}$$

où $l(\overline{v})$ désigne la longueur du vecteur $\overline{v}$ et où $\subset_1$ désigne le fait d'être un sous-vecteur ne contenant que des "1" d'un vecteur, *i.e.*

$$v_1 \subset_1 v_2 \Leftrightarrow \exists j : v_1(i) = v_2(i+j), \ \forall i. \tag{5.59}$$

L'évolution de la fitness (uniquement la composante définie par 5.58) du meilleur individu de chaque génération est tracée sur la figure 5.16 pour la version GAWN avec le 1-crossover qui inclut un terme de pénalité dans la fitness pour la fréquence (courbe bleue) et pour la version qui utilise la nouvelle définition du crossover (courbe rouge). On voit clairement que les performances du GA sont fortement augmentées par l'utilisation de ce nouveau crossover sur le problème considéré. Évidemment, lorsqu'on considère un problème où il n'est pas important de garder les fréquences constantes, l'utilisation de la nouvelle définition du crossover n'augmentera pas forcément les performances du GA.

Enfin, une dernière amélioration que nous avons apportée est l'*hybridation avec une méthode de recherche locale*. En effet, on remarque souvent qu'à partir d'un certain moment, l'évolution de l'algorithme stagne et ne repose plus que sur les mutations effectuées. Ce phénomène peut être observé sur la figure 5.16 par les longs plateaux de stagnation de la fitness. L'idée est d'effectuer *un* pas d'une recherche locale sur le meilleur individu de la population lorsque la recherche du GA stagne depuis un certain nombre de générations. Si la recherche locale trouve une amélioration dans l'individu,

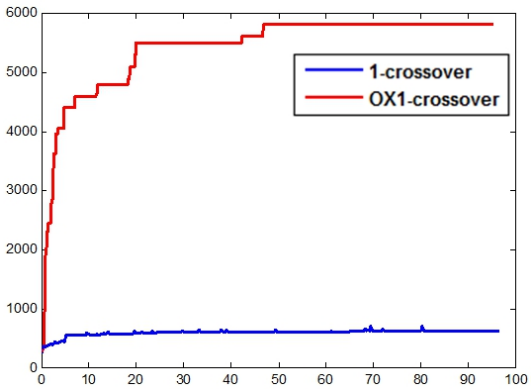


FIGURE 5.16 – Comparaison de l'évolution de la fitness du meilleur individu entre le 1-crossover et la nouvelle définition issue du OX1.

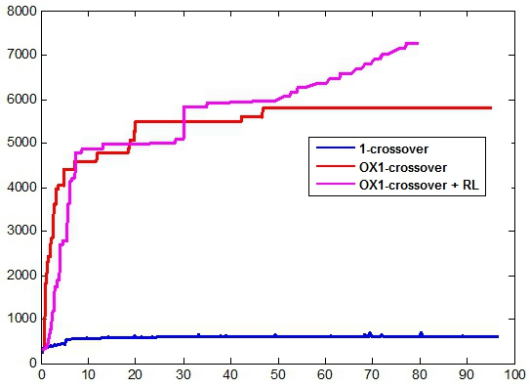


FIGURE 5.17 – Comparaison de l'évolution de la fitness du meilleur individu entre le 1-crossover (bleu), la nouvelle définition issue du OX1 (rouge) ainsi que la version où l'on rajoute une recherche locale (rose).

cela permet généralement de relancer le GA car les changements apportés vont pouvoir se propager dans la population et améliorer éventuellement la qualité d'autres individus, et ainsi de suite. D'un autre côté, si la recherche locale n'arrive pas à déterminer un meilleur individu, cela signifie que l'on a un maximum local pour la recherche considérée. Le fait de pouvoir dire que les solutions trouvées sont V-optimales, où V est le type de voisinage défini par la méthode, garantit un certain niveau de qualité des solutions. Nous avons choisi ici le voisinage de l'ensemble des 2-permutations

$$V(b) = \{\bar{b} : \exists i, j : i \neq j : \bar{b}_i = b_j, \bar{b}_i = b_j \text{ et } \bar{b}_k = b_k \ \forall k \notin \{i, j\}\}. \quad (5.60)$$

Les solutions renvoyées par le GA seront donc *2-optimales*. Pour illustrer les performances du GA avec une recherche locale, nous avons appliqué les 3 algorithmes (GAWN, GAWN + OX1 et GAWN + OX1 + recherche locale) sur le problème de la plus longue chaîne de "1". Les résultats sont représentés sur la figure 5.17. On peut voir que la recherche locale a bien l'effet escompté puisqu'elle permet de relancer l'algorithme lorsque celui-ci stagne depuis quelques générations.

Nous avons même basé notre critère d'arrêt sur le résultat de la recherche locale : on arrête l'exécution du GA après 3 recherches locales infructueuses d'affilée, en précisant que les recherches locales successives sont espacées d'un certain nombre de générations pour permettre au GA d'éventuellement évoluer grâce à ses propres mécanismes de variation. Cela permet d'avoir un critère d'arrêt plus pratique que le critère sur le nombre de générations, car il est difficile de prédire à l'avance combien de générations il faut pour converger, alors que la condition qui dépend de la recherche locale constitue un critère de convergence V-optimale.

5.6 Résultats

Présentons maintenant les résultats des simulations numériques effectuées. Dans toutes nos simulations, nous considérons des expériences de $t = 480$ secondes et un temps d'acquisition TR de 2 secondes. La taille des strings de S sera donc $n = \frac{480}{2} = 240$. Nous considérons des expériences à deux types de stimuli de fréquences égales, sans phases de repos. Pour la définition de l'efficacité (5.35), nous nous restreignons au contraste différentiel $c = (1 - 1)$. La taille de la population a quant à elle été fixée à $m = 50$ individus.

Dans la section 5.6.1 on présente les résultats de l'optimisation mono-objectif, c'est-à-dire l'optimisation par rapport à chacun des critères F_1, F_2 et F_4 pris séparément (puisqu'il n'y a pas besoin de considérer F_3 le terme de fréquence grâce à l'utilisation du crossover OX1). Ensuite, on présente les résultats de l'optimisation où l'on rajoute le critère de la robustesse aux deux premiers critères dans la section 5.6.2. Enfin, dans la section 5.6.3, on donne les résultats de l'optimisation où les objectifs F_1, F_2 et F_4 sont tous trois considérés.

5.6.1 Optimisation mono-objectif

Efficacité

Considérons tout d'abord l'optimisation de l'efficacité pour le contraste $c = (1, -1)$. Dans un premier temps, considérons que nous n'appliquons pas de filtre temporel (cf. section 5.3.2). Le résultat de l'exécution de l'algorithme est donné à la figure 5.18. Il s'agit d'une expérience par blocs (cf. section 5.2), où chaque bloc contient une dizaine de stimuli. Ce résultat confirme les résultats présentés dans [Wager and Nichols, 2003], où la même simulation est faite. La différence est que les améliorations apportées à l'algorithme permettent de converger plus vite vers la solution. Notons qu'en général, lorsqu'auncun filtre temporel n'est appliqué, les expériences par blocs sont en général très bonnes lorsque les blocs sont assez grands. Comme le montre la figure 5.19, toutes les expériences par blocs de longueur au moins égale à 10 sont presque aussi performantes (entre 95% et 100% de la valeur optimale).

Cependant, les expériences qui font intervenir des longs blocs ne sont généralement pas efficaces en pratique, à cause de la corrélation temporelle. C'est pourquoi nous avons considéré ensuite le cas d'un modèle où le bruit est hautement auto-corrélé. Le modèle pour le bruit est le même que celui utilisé dans [Wager and Nichols, 2003], qui correspond à un modèle qui se veut refléter la corrélation typique de données réelles. Le résultat trouvé avec ce modèle est donné sur la figure 5.20. On peut voir qu'on a toujours une expérience par blocs, mais que la longueur des blocs a diminué (environ 5 stimuli par bloc). Le fait d'avoir une expérience avec une plus haute fréquence

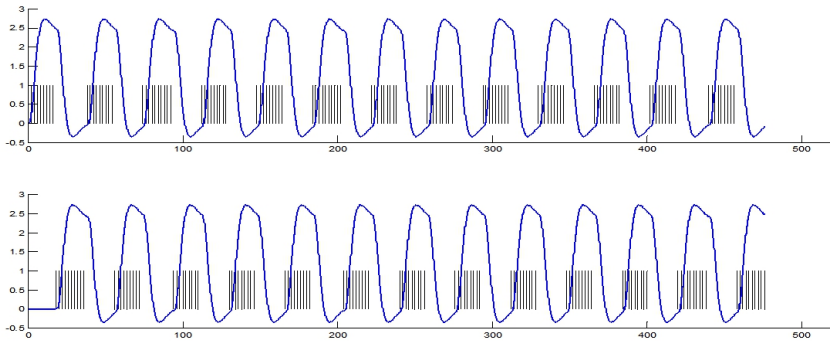


FIGURE 5.18 – Expérience optimale pour le critère de l'efficacité définie par (5.35), pour chaque type de stimulus. Aucun filtre anti-corrélation n'est considéré.

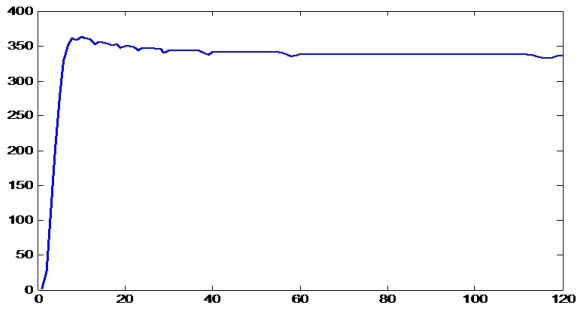


FIGURE 5.19 – Graphe de l'efficacité en fonction de la longueur des blocs des expériences par blocs. Aucun filtre anti-corrélation n'est considéré.

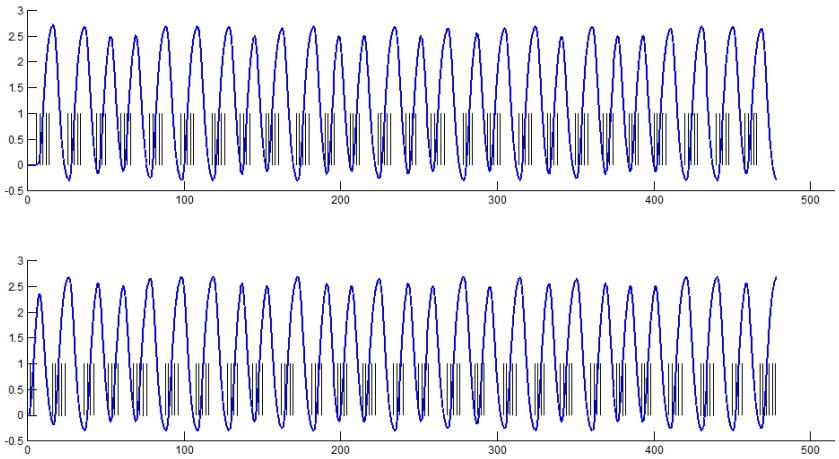


FIGURE 5.20 – Expérience optimale pour le critère de l'efficacité pour chaque type de stimulus, avec filtre anti-corrélation.

permet en fait d'éviter que la fréquence du signal ne se confonde avec le signal du bruit résiduel qui se situe principalement dans les basses fréquences.

Contrebalancement

Considérons maintenant le critère du contrebalancement défini par (5.40). On considère d'abord un contrebalancement d'ordre 1. La représentation de l'expérience trouvée est donnée sur la figure 5.21. Il s'agit d'une expérience d'apparence aléatoire. Ce n'est pas surprenant car intuitivement on se rend bien compte qu'une expérience aléatoire sera difficile à prédire. On remarque aussi que l'algorithme met très peu de temps à atteindre un optimum (3 ou 4 générations). Ce n'est pas étonnant puisque la population initiale est composée d'un ensemble d'expériences déterminées de manière aléatoire, et donc potentiellement bien contrebalancées.

En lançant l'algorithme avec des ordres plus grands, on obtient de nouveau des expériences qui ont ce même aspect aléatoire. La différence est que l'algorithme prend un peu plus de temps à converger. Cela s'explique par le fait que les contrebalancements d'ordres supérieurs sont plus "exigeants", et demandent de vérifier de bonnes proportions dans les successions de stimuli sur plusieurs ordres, ce que ne satisfont pas forcément les expériences déterminées de manière aléatoire, ou pas aussi facilement que pour l'ordre 1.

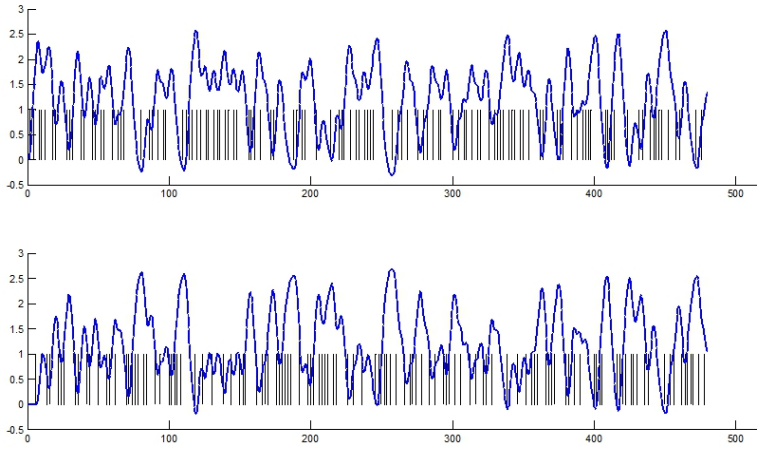


FIGURE 5.21 – Exemple d'expérience optimale pour le contrebalancement d'ordre 1.

Robustesse

Considérons maintenant le critère de la robustesse. Le résultat obtenu à l'issue de l'exécution du GA est assez surprenant et est représenté sur la figure 5.22. Il s'agit de l'expérience qui consiste à présenter un stimulus de chaque type de manière parfaitement alternée. Ce résultat est un peu surprenant car a priori, on serait tenté de croire que les expériences les plus robustes sont les expériences en blocs. En effet, quand les stimuli sont regroupés en blocs, le signal forme lui aussi des blocs d'une certaine amplitude. Les signaux pour deux fonctions HRF différentes ne correspondront donc pas lors des phases ascendantes et descendantes de chaque bloc, mais auront la même amplitude sur la partie centrale du bloc, comme illustré sur la figure 5.9. Ceci dit, l'alternance parfaite peut être vue d'une certaine manière comme une expérience par blocs particulière. En effet, les stimuli pour chaque type peuvent être vus comme appartenant à un grand bloc tout au long de l'expérience, avec un ISI=2TR entre stimuli de même type. Le signal sera donc presque constant, avec de petites oscillations, pendant toute la session, et ce pour les deux types de stimuli, ce qui permettra d'avoir une bonne robustesse.

Cependant, lorsqu'on augmente, même légèrement, la valeur de l'ISI, le résultat précédent n'est plus vrai, et l'expérience par alternance parfaite devient même très mauvaise pour le critère de la robustesse. En effet, à partir d'un ISI de 2.1, c'est l'expérience représentée sur la figure 5.23 qui devient optimale. Cette expérience correspond à soumettre le premier type de stimulus pendant la première moitié de la session, et ensuite soumettre le deuxième type de stimulus pendant le reste de la session. De manière générale, on peut observer que les expériences les plus robustes sont les expériences par blocs.

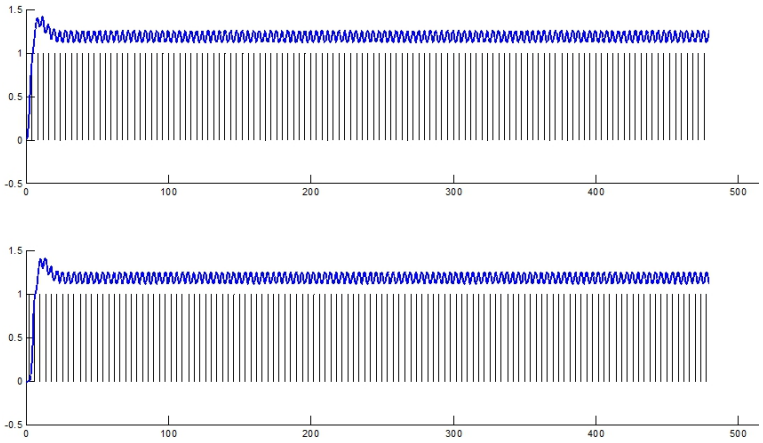


FIGURE 5.22 – Expérience optimale pour le critère de la robustesse.

Un représentation de la robustesse en fonction de la longueur des blocs (en nombre de stimuli par bloc) est faite sur la figure 5.24. On peut voir que plus la longueur des blocs est grande, plus l'expérience est robuste, exception faite de l'expérience par alternance parfaite.

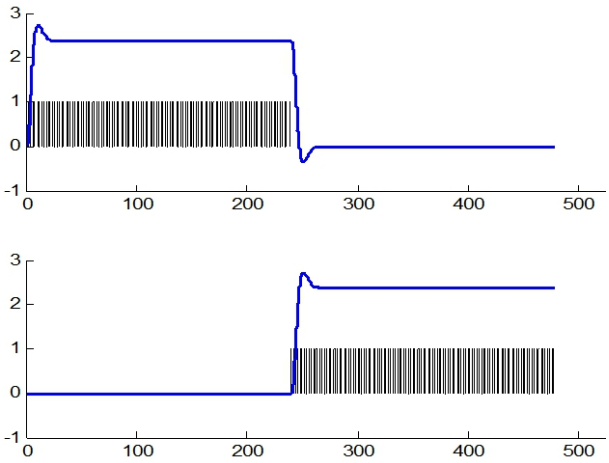


FIGURE 5.23 – Expérience optimale pour la robustesse, après l'expérience par alternance parfaite.

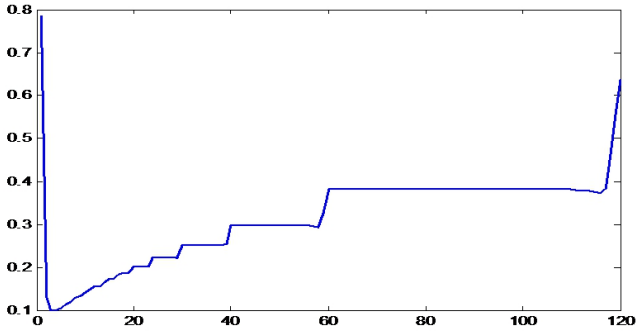


FIGURE 5.24 – Graphe de la robustesse en fonction de la longueur des blocs (en nombre de stimuli par bloc) des expériences par blocs.

Remarquons cependant que les deux expériences optimales présentées correspondent aux expériences de contrebalancement minimum. On peut donc déjà prédire qu'il risque d'y avoir conflit entre ces deux critères, puisque le critère de la robustesse tente en quelque sorte de "créer des blocs", alors que le contrebalancement essaye de "rendre l'expérience aléatoire".

5.6.2 Optimisation robuste

Dans cette section, on tente de coupler les optimisations par rapport à l'efficacité et au contrebalancement avec la robustesse.

Efficacité et robustesse

Considérons d'abord le critère de l'efficacité. La fonction objectif considérée est donc de la forme (5.50), où l'on considère le poids pour le contrebalancement $w_2 = 0$. Dans un premier temps, on considère un modèle d'erreur sans auto-corrélation temporelle.

Le résultat de l'optimisation avec des poids identiques $(w_1, w_4) = (0.5, 0.5)$ est représenté sur la figure 5.23. Il s'agit de l'expérience en deux blocs (de 120 stimuli chacun), qui constitue une expérience qui est (très) bonne pour les deux critères. En effet, on peut exprimer le *degré d'optimalité* d'une expérience comme la valeur de la fitness *standardisée* (5.49) pour chaque critère. Dans ce cas-ci, l'expérience trouvée est optimale à 77% environ pour la robustesse, et à 95% pour l'efficacité.

Qu'en est-il pour des poids différents ? Lorsqu'on fait varier légèrement les poids, aucun changement n'apparaît dans le résultat trouvé. Seul le temps d'exécution change d'une fois à l'autre. Ce n'est que lorsqu'on considère des valeurs extrêmes, à partir de $(w_1, w_4) = (0.95, 0.05)$ jusque la valeur frontière $(w_1, w_4) = (1, 0)$, que l'on observe des changements dans le comportement de l'algorithme. On converge dans ce cas vers l'expérience par blocs de 10 stimuli, représenté sur la figure 5.18. Celle-ci constitue le maximum pour l'efficacité prise comme critère unique, mais amène à une robustesse très mauvaise de 0.08%.

Pour expliquer ce comportement, on plote les points (F_1, F_4) pour l'ensemble des expériences par blocs, où l'on fait varier les longueurs des blocs. La représentation du nuage de points obtenus est faite sur la figure 5.25. Comme nous l'avons déjà fait remarquer auparavant, l'optimisation multi-objectifs par combinaison convexe permet d'atteindre les sommets de l'*enveloppe convexe* de l'ensemble des points (F_1, F_4). Or dans notre cas, cette enveloppe convexe n'est composée que de trois points : les blocs de 1 et de 10 stimuli, *i.e.* les deux points optimaux pour l'optimisation mono-objectif pour l'efficacité et

la robustesse respectivement, ainsi que l'expérience par blocs de 120 stimuli. La convergence ne peut donc théoriquement se faire que vers l'un de ces trois points. Cependant les points extrêmes ne constituent pas de bons compromis, ils ne sont très bons que pour l'un des deux critères, et très mauvais pour l'autre. Il faudra donc mettre beaucoup de poids sur l'un ou sur l'autre critère pour atteindre ceux-ci.

Pour savoir à partir de quelles valeurs des poids on va atteindre l'un ou l'autre point, on peut calculer la pente de la droite reliant deux sommets de l'enveloppe convexe. En effet, on peut réécrire le problème de maximisation $\max F = w_1 F_1 + w_4 F_4$ comme

$$F_4 = -\frac{w_1}{w_4} F_1 + \frac{1}{w_4} F, \tag{5.61}$$

qui est l'équation d'une droite de coefficient angulaire $-\frac{w_1}{w_4}$ et d'ordonnée à l'origine $\frac{1}{w_4}F$. Comme on veut maximiser F, le problème d'optimisation revient à trouver le point pour lequel la droite (5.61) est la plus "haute" possible. Le sommet particulier de l'enveloppe convexe pour lequel l'ordonnée à l'origine de cette droite sera maximum dépend donc du coefficient angulaire de la droite. Pour notre problème, on peut calculer que l'expérience par alternance devrait être atteinte pour $w_1 \in [0, 0.19]$ (et donc $w_2 \in [0.81, 1]$ puisque $w_1 + w_2 = 1$), l'expérience par blocs de 120 pour des valeurs $w_1 \in [0.19, 0.93]$ ($w_2 \in [0.07, 0.81]$), et l'expérience par blocs de 10 pour $w_1 \in [0.93, 1]$ ($w_1 \in [0, 0.07]$).

Notons que ces règles ne sont pas toujours respectées en pratique lorsqu'on lance l'algorithme, surtout pour des valeurs des poids proches des valeurs de transition. On obtient parfois des expériences qui ne sont pas par blocs réguliers et qui constituent des maxima locaux desquels le GA ne sait pas s'échapper, et il faut alors plusieurs runs avant d'atteindre une expérience par blocs.

Considérons maintenant le problème avec l'erreur qui est auto-corrélée. On utilise pour cela le même modèle d'erreur que dans la section 5.6.1. Comme précédemment, on considère tout d'abord des poids égaux $(w_1, w_4) = (0.5, 0.5)$. L'expérience trouvée est l'expérience par blocs d'environ 5 stimuli par bloc, et est représentée sur la figure 5.20. C'est la même expérience que celle trouvée pour l'optimisation par rapport à l'efficacité uniquement. Cependant, comme on peut voir sur la figure 5.24, celle-ci est l'une des pires expériences pour la robustesse.

Tentons de faire varier les poids (w_1, w_4) pour essayer d'obtenir une expérience plus robuste. Cependant, la variation des poids n'a pas d'effet sur le résultat. Pour tenter d'en comprendre la raison, traçons de nouveau le nuage

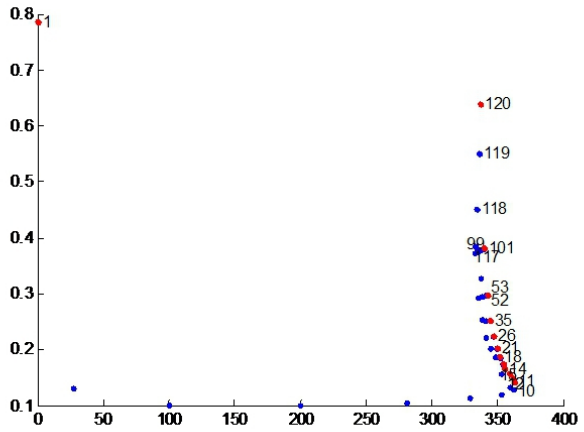


FIGURE 5.25 – Nuage des points (F_1, F_4) formés des objectifs de l'efficacité (en abscisse) et de la robustesse (en ordonnée), pour l'ensemble des expériences par blocs. Aucun filtre temporel n'est appliqué. Chaque point est labellisé par sa longueur de blocs. Les points en rouge forment la *frontière de Pareto*.

des points (F_1, F_4) pour les expériences par blocs. Le résultat est donné sur la figure 5.26. On remarque que les points sont agencés de manière particulière, en formant une courbe du type $1/x$. Cela reflète l'*incompatibilité* entre les deux critères, puisque l'amélioration d'un des critères engendre une forte réduction de la qualité de l'autre critère. Mais cela a pour conséquence aussi que l'enveloppe convexe n'est formée que des deux points optimaux pour chacun des critères pris séparément, et l'optimisation multi-objectifs est incapable d'atteindre des expériences qui forment un *compromis* entre les deux critères. Pourtant, il serait sans doute intéressant de pouvoir atteindre des points qui se situent entre les deux extrêmes, comme par exemple l'expérience par blocs de longueur 26.

Une notion plus large pour l'optimisation multi-objectifs et qui permet de prendre en compte ces candidats est la notion de *front de Pareto*. Pour cela, on définit d'abord la notion de *dominance* d'un point sur un autre point. Soient x et y deux points de $\mathbb{R}^n$. On dit que x *domine* y, et on note $x \gg y$, si le vecteur x est plus grand que le vecteur y *pour chacune de ses composantes*, *i.e.*

$$x \gg y \Leftrightarrow \forall i = 1, \ldots, n : \; x_i > y_i. \tag{5.62}$$

On peut maintenant définir le front de Pareto d'un ensemble donné comme l'ensemble des points qui ne sont dominés par aucun autre point. Le front de Pareto désigne en fait l'ensemble des points qui ne sont pas comparables

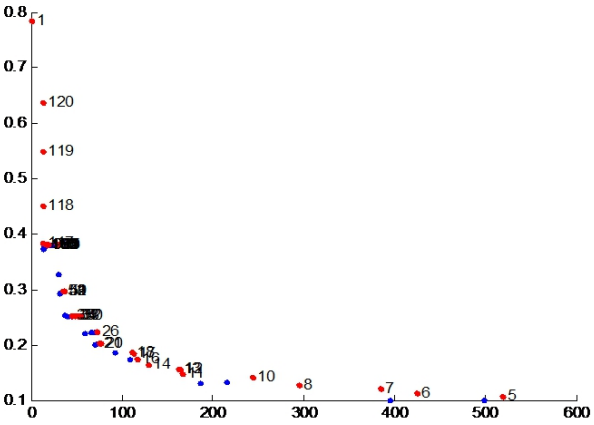


FIGURE 5.26 – Nuage des points (F_1, F_4) formés des objectifs de l'efficacité (en abscisse) et de la robustesse (en ordonnée), pour l'ensemble des expériences par blocs. Un filtre temporel est appliqué pour le calcul de l'efficacité. Chaque point est labellisé par sa longueur de blocs. Les points en rouge forment la *frontière de Pareto*.

par la notion de dominance, et est donc plus large que la notion d'enveloppe convexe. Pour chacun des nuages de points 5.25 et 5.26, nous avons tracé celui-ci en rouge. On voit que le front de Pareto permet de mettre en avant un ensemble de points qui ne sont pas accessible par le GA. Il serait sans doute plus intéressant d'utiliser une méthode de détection du front de Pareto plutôt qu'une méthode d'optimisation par combinaison convexe des objectifs. En effet, cela permettrait d'abord d'atteindre les points qui ne sont pas accessibles avec le GA et qui pourraient pourtant constituer des compromis intéressants entre les différents objectifs. D'autre part cela permettrait de ne pas faire le choix de l'importance de chacun des critères *a priori* par la définition des poids w_i, mais *a posteriori* en toute connaissance d'un ensemble de solutions. Plusieurs méthodes ont été développées pour traduire cette idée, citons par exemple le MOGA (Multi-Objective Genetic Algorithm, [Fonseca and Fleming, 1993]) ou le NSGA (Nondominated Sorting Genetic Algorithm, [Srinivas and Deb, 1995]). Ces méthodes n'ont pas été implémentées dans ce travail, mais il serait très intéressant de le faire dans un travail ultérieur.

Pour revenir au problème de l'optimisation selon les critères d'efficacité et de robustesse, nous ne donnerons donc pas ici de solution en particulier, mais l'ensemble des solutions du front du Pareto. Remarquons que nous avons fait dans cette section l'hypothèse implicite que les expériences par blocs étaient optimales pour l'efficacité et la robustesse. Cependant, il se pourrait que certaines expériences intermédiaires, avec des longueurs de blocs variables

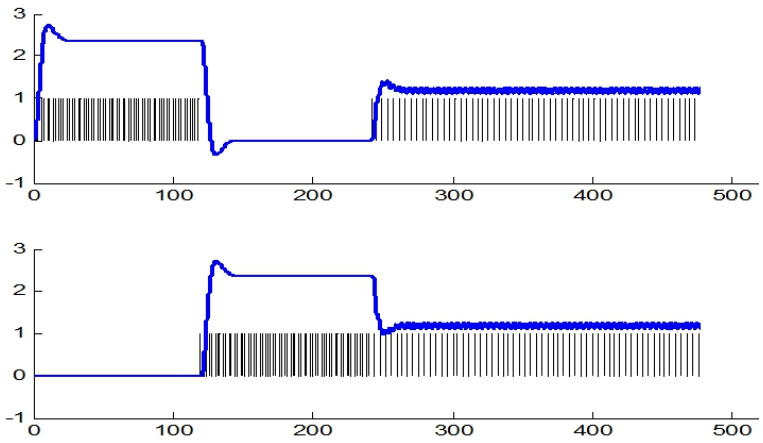


FIGURE 5.27 – Expérience optimale pour les critères du contrebalancement (d'ordre 1) et de la robustesse.

par exemple, soit aussi bonnes que d'autres expériences par blocs. Nous avons donc donné ici une idée de la structure des solutions, mais il serait sans doute préférable d'utiliser d'autres méthodes pour préciser les résultats obtenus. Remarquons déjà à ce point que des difficultés similaires risquent d'être rencontrées pour l'optimisation de l'ensemble des trois objectifs F_1, F_2 et F_4 (cf. section 5.6.3).

Contrebalancement et robustesse

Penchons nous maintenant sur l'optimisation des critères du contrebalancement et de la robustesse. Considérons tout d'abord un contrebalancement d'ordre 1. L'expérience optimale est représentée sur la figure 5.27. Cette expérience est un mix entre l'expérience en deux blocs et l'expérience par alternance parfaite. Le fait que cette expérience soit robuste n'est pas surprenant, parce qu'elle couple deux expériences robustes. Ce qui est un peu plus surprenant est que cette expérience constitue une expérience optimale pour le contrebalancement d'ordre 1. En effet, les blocs homogènes contiennent chacun $n/4$ stimuli du même type qui se succèdent, et le bloc d'alternance de la deuxième partie de l'expérience contient $n/2$ sitmuli pour les deux types confondus, ce qui fait $n/4$ stimuli de chaque type qui succèdent à des stimuli de l'autre type. D'un point de vue *global*, le contrebalancement d'ordre 1 est donc parfait.

Cependant, cette expérience ne permet pas d'éviter la prédictibilité car localement le contrebalancement est très mauvais. Cela peut être fait en considérant un ordre supérieur pour le contrebalancement. De plus, rappelons qu'il est préférable d'éviter l'expérience par alternance parfaite car

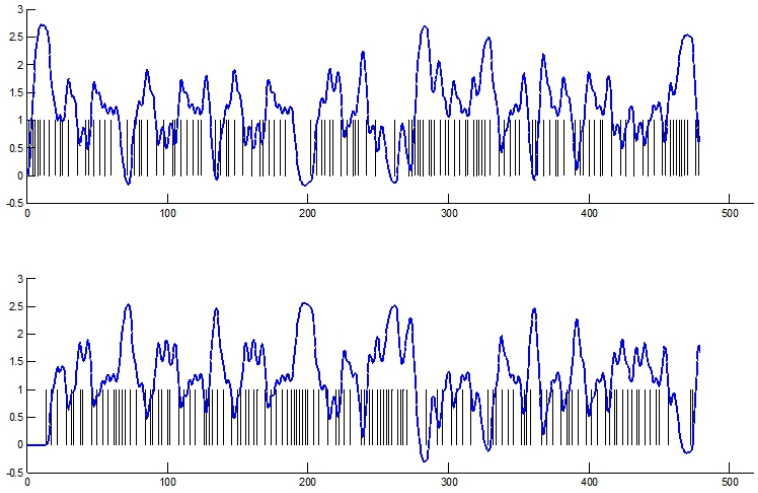


FIGURE 5.28 – Expérience optimale pour les critères du contrebalancement (d'ordre 3) et de la robustesse avec des poids identiques pour les deux critères.

sa robustesse dépend sensiblement de l'ISI choisi. Nous avons donc choisi de considérer un contrebalancement d'ordre 3. Le résultat trouvé avec des poids identiques est donné sur la figure 5.28. Cette expérience semble bien respecter le contrebalancement cette fois, mais est très faible pour la robustesse (moins de 5% de l'optimum). Il semblerait donc que le GA ait convergé vers un point qui n'est optimum que pour un seul des critères, en l'occurrence le contrebalancement.

Cependant, lorsqu'on rajoute du poids sur la robustesse il semblerait qu'on arrive à atteindre des solutions de compromis. Par exemple la figure 5.29 montre une expérience obtenue à l'issue d'une exécution de l'algorithme. On peut voir que le terme de robustesse a permis de "créer des blocs" dans une expérience aléatoire. Les blocs créés ont été entourés en rouge sur la figure. L'idée pour concilier les deux critères est donc sans doute de créer des expériences intermédiaires entre les expériences aléatoires et les expériences par blocs. Les blocs permettent alors d'avoir des zones robustes, et le reste de l'expérience permet d'équilibrer les fréquences de succession pour chaque type de stimulus afin d'avoir un contrebalancement global correct.

5.6.3 Optimisation multi-objectifs

Dans cette section on considère l'optimisation par rapport aux 3 objectifs : l'efficacité, le contrebalancement et la robustesse. On considère un modèle de l'erreur avec auto-corrélation, et on applique donc un filtre temporel pour l'efficacité. L'ordre maximum pour le contrebalancement est fixé à $R = 3$.

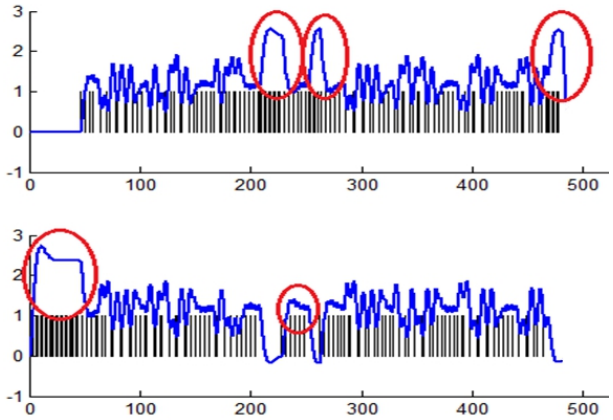


FIGURE 5.29 – Expérience optimale pour les critères du contrebalancement (d'ordre 3) et de la robustesse avec un poids plus grand sur le critère de la robustesse.

On fixe d'abord chacun des poids à $w_i = 1/3$. Le résultat de l'exécution de l'algorithme est donné sur la figure 5.30. Il s'agit d'une expérience qui constitue un compromis entre efficacité ($\sim$45% de la valeur optimale) et contrebalancement ($\sim$40% de la valeur optimale). Cela se confirme par la structure de l'expérience : il s'agit de petits blocs qui ont été un peu "brouillés" de manière à avoir une bonne efficacité tout en restant assez bien contrebalancé pour éviter la prédictibilité. Cependant, la robustesse est très faible avec à peine 5% de la valeur optimale. L'expérience obtenue se situe donc probablement sur une frontière de l'enveloppe convexe (en trois dimensions).

Pour tenter d'avoir une expérience avec une meilleure robustesse, on ajoute du poids sur le critère de robustesse. Par exemple, l'exécution avec les poids $(w_1, w_2, w_4) = (0.2, 0.2, 0.6)$ donne l'expérience représentée sur la figure 5.31. Celle-ci est constituée d'un long bloc de stimuli du premier type dans une première partie de l'expérience, et ensuite les stimuli sont agencés de manière assez désordonnées, avec tout de même 3 blocs de longueur moyenne pour le deuxième type de stimulus. Il s'agit ici d'un compromis entre les critères de robustesse et de contrebalancement. En effet, la robustesse est de $\sim$30%, le contrebalancement de $\sim$35%, mais l'efficacité est de seulement $\sim$5%.

En faisant varier les paramètres de poids, on obtient cependant toujours des solutions dont au moins l'un des objectifs est très mauvais. Il semble donc difficile d'obtenir un compromis pour les trois objectifs simultanément.

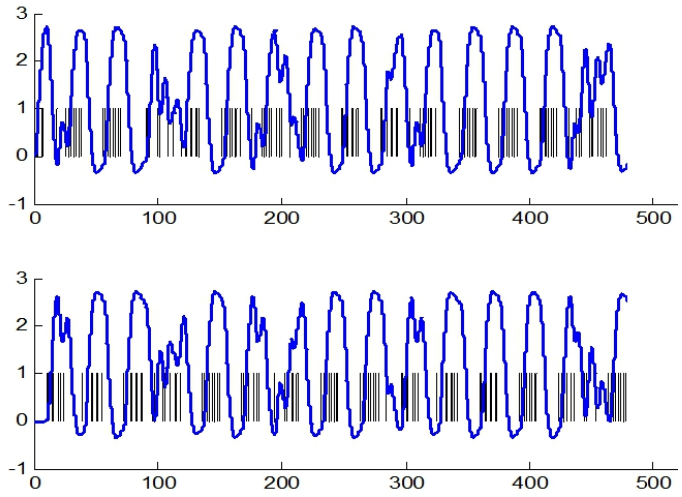


FIGURE 5.30 – Expérience optimale pour l'ensemble des critères d'optimalité, avec des poids identiques.

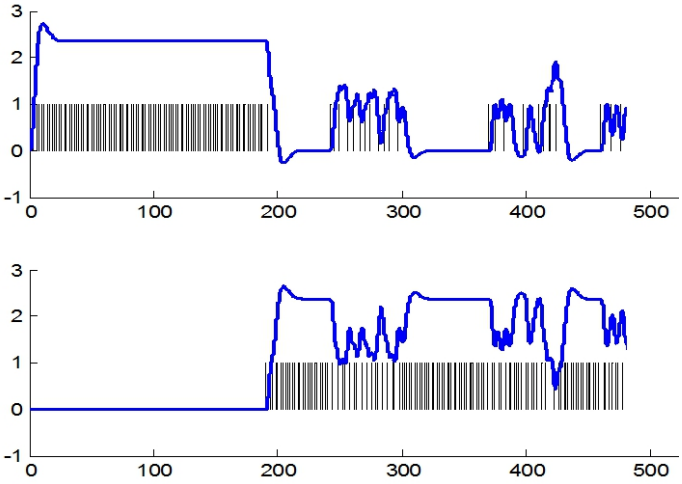


FIGURE 5.31 – Expérience optimale pour l'ensemble des critères d'optimalité, avec les poids $(w_1, w_2, w_4) = (0.2, 0.2, 0.6)$.

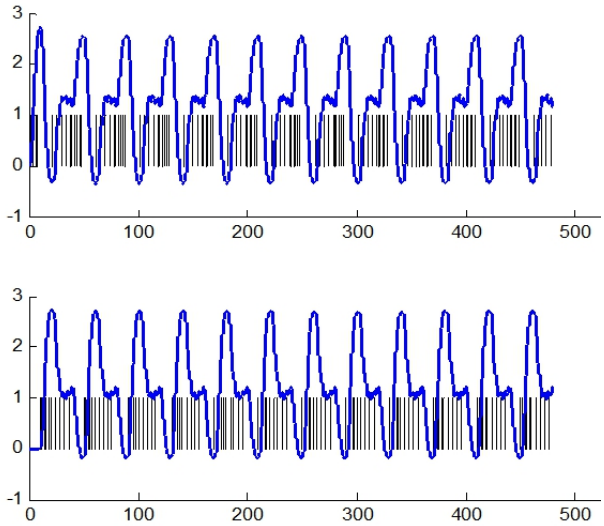


FIGURE 5.32 – Expérience optimale pour l'ensemble des critères d'optimalité, avec la stratégie d'immigration.

De plus, il n'est pas très pratique de devoir lancer un grand nombre de fois l'algorithme avec des poids différents pour espérer obtenir une solution qui soit satisfaisante. Il semblerait donc vraiment opportun de pouvoir utiliser une méthode du type MOGA de manière à pouvoir obtenir un ensemble de solutions dans lequel faire un choix a posteriori.

Pour tenter de trouver une solution de compromis à ce problème, nous avons suivi l'idée proposée dans l'article [Kao et al., 2009], qui consiste à "guider" la recherche par un pas supplémentaire dans le GA, appelé pas d'*immigration*. Il s'agit de rajouter à chaque génération des individus "spéciaux", de manière à "proposer" à l'algorithme certaines structures de solution qu'il peut inclure dans la population si la fitness est assez bonne. Typiquement, les individus immigrants sont des expériences connues pour être de bonne qualité pour certains critères, et donc en particulier des expériences aléatoires (pour le contrebalancement), des expériences par blocs de différentes longueurs (pour l'efficacité et la robustess), ainsi que des combinaisons plus ou moins complexes des deux. Cette technique a permis de trouver l'expérience représentée sur la figure 5.32. Cette expérience constitue cette fois un compromis pour tous les objectifs car aucun n'a de valeur trop petite (tous atteignent au moins 10% de leur valeur optimale). Il s'agit d'un mix entre une expérience avec des petits blocs et une expérience aléatoire. L'efficacité est de ∼27% de sa valeur optimale, le contrebalancement ∼20% et la robustesse ∼10%. Aucune de ces valeurs n'est très élevée, mais c'est inévitable avec des objectifs contradictoires.

Notons cependant que la solution obtenue est le résultat de l'opération de mélange entre deux designs, et n'est pas due directement à l'algorithme. Cependant, une fois inclus dans ce dernier, l'expérience est restée meilleur candidat et l'algorithme s'est arrêté après 3 recherches locales infructueuses (cf. section 5.5.3). Cela garantit donc au moins que l'expérience trouvée est un optimum local. Mais il est clair que des améliorations peuvent être apportées, et qu'une analyse avec des méthodes plus adaptées serait intéressante. Cette recherche a toutefois permis de donner une intuition du type de solution pour l'optimisation multi-objectifs. En effet, la solution doit être d'une certaine manière une combinaison entre une expérience par blocs et une expérience aléatoire.

5.7 Conclusion

Pour conclure cette section, rappelons d'abord brièvement les résultats obtenus. En ce qui concerne l'optimisation mono-objectif, le GA a de bonnes performances, tant au niveau du temps d'exécution que des résultats trouvés, et permet de mettre en avant le type d'expérience optimale pour chaque objectif :

- Pour l'efficacité, sans filtre temporel les expériences optimales sont les expériences par blocs avec au moins 10 stimuli par bloc, alors qu'avec filtre ce sont les expériences par petits blocs (environ 5 stimuli par bloc) qui sont optimales.
- Pour le contrebalancement, de manière générale ce sont les expériences aléatoires qui sont optimales. Notons qu'il est important de considérer un contrebalancement d'ordre assez grand pour éviter des expériences globalement optimales mais qui contiennent quand même une très grande prédictibilité.
- Pour la robustesse, de manière générale ce sont les expériences contenant des blocs qui sont optimales, et plus la longueur des blocs est grande, plus robuste est l'expérience.

Ensuite, nous avons considéré l'optimisation multi-objectif de l'efficacité et du contrebalancement avec la robustesse. Nous avons alors rencontré un peu plus de difficulté à trouver des solutions de compromis, principalement à cause de la stratégie d'optimisation par combinaison convexe qui ne permet pas d'atteindre tous les points du front de Pareto. Nous pouvons cependant retirer de l'analyse effectuée que :

- Pour l'efficacité et la robustesse, sans filtre temporel, la solution optimale est l'expérience en deux blocs de stimuli. Lorsqu'un filtre temporel est appliqué, un compromis doit être trouvé sur la longueur des blocs : plus les blocs sont longs, plus l'expérience est robuste, mais moins elle est efficace, et inversement.

- Pour le contrebalancement et la robustesse, il faut trouver un compromis entre, d'une part, avoir des blocs, et d'autre part, avoir une structure aléatoire qui permette la non-prédictibilité de l'expérience. Typiquement, cela peut être obtenu en dispersant plusieurs petits blocs dans l'expérience, et en complétant celle-ci avec des parties aléatoires pour avoir un équilibre pour le contrebalancement.

Enfin, nous avons considéré les trois critères simultanément. L'optimisation est plus difficile, de nouveau à cause de la limitation de l'optimisation par combinaison convexe des objectifs. Nous avons néanmoins mis en avant un type d'expérience qui pourrait s'avérer être un compromis raisonnable entre tous les objectifs, qui consiste à mélanger de manière plus ou moins complexe une expérience par (petits) blocs avec une expérience aléatoire.

En ce qui concerne l'algorithme génétique, ce chapitre a pu mettre en avant ses limites. En effet, de par sa nature aléatoire, il peut renvoyer des résultats assez différents d'une exécution à l'autre. Il faut donc souvent plusieurs exécutions et une analyse des résultats obtenus avant de pouvoir conclure d'une solution optimale. Cependant, les principales difficultés rencontrées sont plus dues à l'optimisation multi-objectifs qui est limitée par l'approche envisagée qu'à l'algorithme lui-même. Les algorithmes génétiques restent un outil très pratique et très flexibles. Comme nous l'avons déjà mentionné, des adaptations dans la structure des GAs peuvent être faites pour mieux répondre aux difficultés relatives à l'optimisation multi-objectifs. Il serait donc intéressant dans un travail ultérieur d'utiliser de telles méthodes pour le problème d'expérience optimale pour le fMRI.

Notons également que nous n'avons fait aucune étude du temps d'exécution dans cette section. C'est parce que toutes les simulations (sauf pour le contrebalancement) ont pris un temps d'exécution du même ordre : entre 2 heures et 5 heures suivant le nombre de critères à prendre en compte et suivant la capacité de la méthode à trouver rapidement un optimum. Nous n'avons pas poussé l'analyse du temps CPU plus loin car notre priorité était avant tout d'obtenir des solutions de bonne qualité, et que le temps d'exécution a de toute façon déjà été fortement diminué par les améliorations apportées à l'algorithme GAWN. Il serait toutefois intéressant de considérer plus sérieusement le critère du temps CPU que ce que nous l'avons fait.

D'autres perspectives peuvent encore être données. Il serait par exemple aussi intéressant d'inclure l'ISI dans l'optimisation, et de ne pas le garder fixe. Il serait aussi intéressant de considérer d'autres contrastes d'intérêt pour le critère de l'efficacité, de même que d'analyser les résultats pour des expériences impliquant plus que deux types de stimuli différents, ou des ex-

périences contenant des phases de repos. Enfin, la dernière suggestion est celle qui concerne les non-linéarités. Il serait intéressant d'essayer d'en tenir compte, d'une manière ou d'une autre, dans le processus de modélisation ou d'optimisation. Ceci dit, c'est déjà ce qui est fait en partie avec la contrainte du contrebalancement. Cependant, certains ont la conviction que l'optimisation multi-objectifs est un outil utilisé pour pallier à une mauvaise définition du problème (éventuellement parce que nous ne possédons pas les outils mathématiques nécessaires). Il serait donc intéressant de tenter de prendre directement en compte la non-linéarité dans le modèle. De manière analogue, il serait sans doute intéressant d'adopter une approche différente pour le critère de la robustesse. En effet, peut être que la question à se poser n'est pas de comment réduire l'erreur commise par une mauvaise approximation de l'HRF, mais de tenter d'estimer sa forme dans la modélisation...

Références pour le chapitre 5

Trois sources ont été principalement utilisées pour ce chapitre. La première est le livre [Poldrack et al., 2010] d'introduction à l'analyse des données fMRI, dont Thomas Nichols en est co-auteur. Il a servi principalement de source pour fixer le cadre général de l'étude des données fMRI, et concerne donc plus particulièrement les sections 5.1, 5.2 et 5.3. Les informations contenues dans ces sections ont été recoupées avec une deuxième source, la thèse de Maus [Maus, 2011] sur l'optimisation des expériences par événements pour le fMRI. Enfin, les sections 5.4 et 5.5 sont basées sur le papier [Wager and Nichols, 2003] de Wager & Nichols (2003), qui présente l'optimisation des expériences grâce à l'algorithme GAWN.

Soulignons à nouveau que les parties réellement innovatrices de ce chapitre sont la considération de la robustesse dans l'optimisation multi-objectifs, ainsi que les améliorations du GAWN. La section 5.6.1, la deuxième partie de 5.5.3 ainsi que toute la section 5.6 représentent donc la contribution personnelle de ce travail.

Conclusion

Pour conclure ce travail, passons en revue les différents points traités dans ce travail.

Dans le chapitre 1, nous avons fixé le domaine d'application des GAs. Pour ce faire, nous avons d'abord introduit les problèmes d'optimisation combinatoire, en donnant comme exemples le TSP et le problème de coloration des graphes. Nous avons ensuite expliqué que ces problèmes sont généralement difficiles à résoudre, et nous avons donné un aperçu de la raison de cette difficulté, principalement grâce aux notions de complexité des algorithmes et des problèmes. Ensuite, nous avons présenté les différents types de méthodes permettant de résoudre ces problèmes, dont font notamment partie les algorithmes génétiques.

Ensuite, dans le chapitre 2, nous avons présenté les différentes composantes des algorithmes génétiques, avec d'une part la structure algorithmique générale et d'autre part les principales spécifications de cette structure. En particulier, nous avons présenté les stratégies de sélection par roulette, par tournoi et par élitisme, les stratégies de reproduction par $1-$crossover, par N_c-crossover, par crossover uniforme et par shuffle crossover, et enfin les stratégies de mutation par $1-$inversion, par N_M-inversion, par inversion totale, par sélection aléatoire et par inversion aléatoire. Nous avons aussi simulé l'exécution d'un GA sur un exemple basique, ce qui nous a permis d'avoir une première approche du mécanisme de fonctionnement des GAs.

Dans le chapitre 3, nous avons donné une vision un peu plus théorique du fonctionnement des GAs, avec la présentation du célèbre *théorème des schémas* de J. Holland (1975). Nous avons vu que celui-ci ne garantit aucunement la convergence de la méthode, contrairement aux preuves que l'on peut retrouver pour les méthodes classiques, mais qu'il fournit plutôt une justification de son utilisation. Cela nous a permis de comprendre plus en profondeur le mécanisme de fonctionnement des GAs. Nous avons vu notamment que la stratégie sous-jacente utilisée par ceux-ci consiste en fait à augmenter progressivement le nombre de schémas de petite taille et de fitness au-dessus de la moyenne. Nous avons ensuite expliqué l'existence du

parallélisme implicite qui permet aux GAs de faire le lien entre les schémas décrits par le théorème et les individus manipulés en pratique. Finalement, nous avons énoncé l'hypothèse sur laquelle reposent les performances des GAs, qui consiste à supposer que la solution optimale peut être atteinte par des améliorations successives dues aux croisements effectués sur les individus. Même si cette hypothèse est souvent vérifiée, nous avons donnés des exemples où ce n'est pas le cas. Nous avons également souligné l'importance de la représentation des individus pour le respect de cette hypothèse, et donc pour de bonnes performances du GA.

Le chapitre 4 s'intéresse à l'application des GAs sur un problème concret : celui de trouver une stratégie optimale pour un robot qui doit nettoyer une surface jonchée de détritus. La première étape a été de donner la modélisation du problème, par la définition de l'ensemble des solutions admissibles et de la fonction objectif. Ensuite, après avoir expliqué la structure algorithmique choisie et implémentée pour traiter ce problème, nous avons tenté une exécution de l'algorithme avec un jeu de paramètres choisi arbitrairement. Nous nous sommes rendu compte que le choix des paramètres est primordial pour une bonne performance de l'algorithme. C'est pourquoi nous avons alors étudié l'influence des différents paramètres sur les performances du GA, notamment dans le but de mettre en avant un jeu de paramètres qui puissent amener à une solution optimale. Cette analyse a permis d'une part de trouver un jeu de paramètres plus performant que celui qui avait été considéré auparavant, mais a permis également de faire plusieurs constatations par rapport à l'influence des différents opérateurs. De manière globale, il semble important de trouver un équilibre entre la spécification, réalisée par l'opérateur de sélection, et la variation, réalisée par les opérateurs de reproduction et de mutation. Une fois ces constatations faites, nous avons lancé une longue exécution du GA avec les paramètres optimaux trouvés. La solution trouvée est "presque" optimale, mais certains gènes restent sous-optimaux. Cependant, cette stratégie a permis de déduire la (une) stratégie optimale. L'objet principal de cette stratégie réside dans le fait de d'abord se déplacer jusqu'à l'extrémité des groupes de canettes adjacentes *sans les ramasser*, et ensuite faire le chemin inverse en collectant chacune de celles-ci.

Dans le dernier chapitre, nous avons étudié le problème d'expériences (par événements) optimales dans le contexte de l'imagerie médicale, et plus particulièrement dans l'imagerie médicale par résonance magnétique fonctionnelle (fMRI). Pour ce faire, nous avons d'abord développé tout le modèle relatif au traitement des données fMRI. L'idée principale est d'écrire ces données grâce au modèle linéaire général (GLM) comme une combinaison linéaire des réponses prédites pour chaque type de stimulus, dont les coefficients sont à estimer, plus un terme d'erreur aléatoire. Nous avons considéré ensuite plusieurs critères d'optimisation : l'efficacité, le contrebalancement, le

respect des fréquences et enfin la robustesse. Nous avons ensuite donné la spécification de l'algorithme génétique utilisé, qui est une version améliorée de [Wager and Nichols, 2003]. Enfin, nous avons réalisé différentes simulations avec le GA. Nous avons d'abord considéré une optimisation de chaque critère séparément, pour ensuite aborder une optimisation multi-objectifs par combinaison convexe des contraintes. Cependant cette optimisation s'est avérée difficile suite aux faiblesses de la méthode utilisée. En effet, nous nous sommes rendu compte que l'optimisation par combinaison convexe ne permet que d'atteindre des solutions de compromis qui se situe dans l'enveloppe convexe des couples reprenant la fitness des solutions par rapport à chaque critère. Cependant, nous avons pu quand même dégager certaines solutions, et obtenir des informations plus générale sur la structure des solutions. Signalons que nous n'avons réalisé dans ce chapitre qu'une petite partie d'un travail qui pourrait être beaucoup plus conséquent, et que de nombreuses pistes ont été dégagées pour continuer le travail entamé.

Notons que de manière générale, ce mémoire a permis de donner un aperçu assez complet du fonctionnement des GAs, d'une part grâce aux considérations plus théoriques du chapitre 3, et d'autre part grâce aux applications des chapitres 4 et 5. En effet, ces deux aspects sont indispensables pour se faire une idée complète d'une méthode donnée. Les applications ont permis ici de se faire une idée réaliste du fonctionnement effectif des GAs, et d'en saisir les principaux avantages et inconvénients. Nous nous sommes rendu compte notamment de l'importance et de la difficulté de la paramétrisation des GAs pour obtenir de bons résultats, en évitant par exemple le problème de la convergence prématurée. De plus, la convergence dépend également de la représentation et de la fitness utilisées. Ce sont tous des aspects qui rendent parfois difficile l'utilisation des GAs. Par contre, ceux-ci ont des avantages indéniables. Un des avantages les plus importants est sans doute le domaine d'application très large de la méthode. En effet, nous avons traité grâce à celle-ci deux problèmes complètement différents et qui ne possèdent aucun lien a priori. Les GAs sont en ce sens très flexibles puisqu'ils ne font aucune hypothèse sur la forme du problème, et ne fait appel à aucune information autre que la fitness.

Terminons par le fait qu'à la fin de chaque chapitre, des pistes ont souvent été données pour un approfondissement de ce qui y a été étudié. Nous n'allons pas redonner la liste des suggestions déjà faites. Donnons à la place quelques perspectives générales, qui ne concernent pas forcément un domaine d'application particulier.

Tout d'abord, il serait intéressant d'approfondir la partie théorique, en étudiant par exemple des résultats tels que ceux présentés par [Cerf, 1994]. En effet, une bonne connaissance théorique permet souvent de comprendre

profondément les mécanismes de fonctionnement des objets étudiés. Cela permettrait sans doute de mieux comprendre les problèmes propres à l'utilisation des GAs, et peut être de savoir mieux y faire face, en proposant des solutions plus "intelligentes", motivées par une connaissance plus profonde du problème.

Par ailleurs, nous n'avons pas envisagé la comparaison des GAs avec d'autres méta-heuristiques, telles que la recherche tabou ou la méthode du recuit simulé. Il serait donc intéressant de comparer les avantages et inconvénients de chacune des méthodes, et de voir notamment si ceux-ci sont compétitifs par rapport aux méthodes comparables.

Enfin, les GAs sont des méthodes qui de par leur grande flexibilité sont beaucoup utilisées dans le domaine de l'industrie. Il serait intéressant de voir dans quelles mesures ces méthodes sont utilisées, et comment l'algorithme est adapté pour répondre aux demandes particulières des différents secteurs d'applications, comme par exemple l'optimisation sous contraintes, l'optimisation multi-objectifs ou l'optimisation à variables mixtes.

Bibliographie

[Aguirre et al., 1998] Aguirre, G. K., Zarahn, E., and D'Esposito, M. (1998). The variability of human, bold hemodynamic responses. *NeuroImage*, 8 :360–369.

[Bandetti and Cox, 2001] Bandetti, P. A. and Cox, R. W. (2001). Event-related fmri contrast when using constant interstimulus interval : Theory and experiment. *Magnetic Resonance in Medicine*, 43 :540–548.

[Banzhaf, 1990] Banzhaf, W. (1990). The "molecular" traveling salesman. *Biological Cybernetics*, 64 :7–14.

[Bertsimas and Tsitsiklis, 1993] Bertsimas, D. and Tsitsiklis, J. (1993). Simulated annealing. *Statistical Science*, 8 (1) :10–15.

[Bodenhofer, 2002] Bodenhofer, U. (2001-2002). *Genetic Algorithm : Theory and Applications (Lecture Notes, second edition)*. Johannes Kepler Universität, Linz, Austria.

[Buckner et al., 1996] Buckner, R. L., Bandettini, P. A., OŠCraven, K. M., Savoy, R. L., Petersen, S. E., and Rosen, M. E. R. B. R. (1996). Detection of cortical activation during averaged single trials of a cognitive task using functional magnetic resonance imaging. *Proceedings of the National Academy of Sciences, USA*, 93 :14878–14883.

[Carletti, 2009] Carletti, T. (Année académique 2008-2009). *Syllabus du cours d'"Algorithmique Mathématique"*. Facultés Universitaires Notre-Dame de la Paix, Namur.

[Caruana and Schaffer, 1988] Caruana, R. A. and Schaffer, D. J. (1988). Representation and hidden bias : Gray vs. binary coding for genetic algorithms. In *Proc. 5th Int. Conf. Machine Learning*, pages 153–161.

[Cerf, 1994] Cerf, R. (1994). *Théorie asymptotique des algorithmes génétiques*. PhD thesis, Université de Montpellier, France.

[Cormen et al., 2009] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms : Third Edition*. MIT press.

[Dale and Buckner, 1997] Dale, A. M. and Buckner, R. L. (1997). Selective averaging of rapidly presentd individual trials using fmri. *Human Brain Mapping*, 5 :329–340.

[Davis, 1985] Davis, L. (1985). Applying adaptive algorithms to epistatic domains. In *Proc. 2nd Int. Conf. on Genetic Algorithms and Their Applications*, pages 162–164. Cambridge, MA.

[Dréo et al., 2003] Dréo, J., Pétrowski, A., Siarry, P., and Éric Taillard (2003). *Métaheuristiques pour l'optimisation difficile*. Eyrolles.

[Fonseca and Fleming, 1993] Fonseca, C. M. and Fleming, P. J. (1993). Genetic algorithms for multiobjective optimization : Formulation, discussion and generalization. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 416–423. Forrest, Ed. San Mateo, CA : Morgan Kauffman.

[Fournier, 2009] Fournier, J. (2009). *Graph Theory and Applications : with exercises and problems*. ISTE.

[Friston et al., 2000] Friston, K., Josephs, O., Zarahn, O., Holmes, A., Rouquette, S., and Poline, J. (2000). To smooth or not to smooth : Bias and efficiency in fmri time-series analysis. *NeuroImage*, 12 (2) :196–208.

[Friston et al., 1999] Friston, K., Zarahn, E., Josephs, O., Henson, R. N., and Dale, A. M. (1999). Stochastic designs in event-related fmri. *NeuroImage*, 10 (5) :607–619.

[Geyer-Schulz, 1995] Geyer-Schulz, A. (1995). *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning*. Physica-Verlag.

[Glover, 1989] Glover, F. (1989). Tabu search — part 1. *ORSA Journal on Computing*, 1 (3).

[Goldberg and Lingle, 1985] Goldberg, D. and Lingle, J. (1985). Alleles, loci and the travelling salesman problem. In *Proc. Int. Conf. Genetic Algorithms and Their Applications*, pages 154–159. Pittburgh, PA.

[Goldberg, 1989] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA.

[Graybill, 1961] Graybill, F. (1961). *An introduction to linear statistical models*. McGraw-Hill Book Company, Inc.

[Hardy, 2008] Hardy, A. (Année académique 2007-2008). *Syllabus du cours de "Statistiques"*. Facultés Universitaires Notre-Dame de la Paix, Namur.

[Holland, 1992] Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems*. First MIT Press ed. The MIT Press, Cambridge, MA. First edition : University of Michigan Press, 1975.

[Hoos and Stützle, 2005] Hoos, H. H. and Stützle, T. (2005). *Stochastic local search : foundations and applications*. Elsevier.

[Kao et al., 2009] Kao, M.-H., Mandal, A., Lazar, N., and Stufken, J. (2009). Multi-objective optimal experimental designs for event-related fmri studies. *NeuroImage*, 44 :849–856.

[Lambert, 2006] Lambert, T. (2006). *Hybridation de méthodes complètes et incomplètes pour la résolution de CSP.* PhD thesis, École doctorale Sciences et Technologies de l'Information et des Matériaux de Nantes, France.

[Larrafiaga et al., 1996] Larrafiaga, P., Kuijpers, C. M. H., Murga, R. H., and Yurramendi, Y. (1996). Learning bayesian network structures by searching for the best ordering with genetic algorithms. *IEEE transactions on systems, man, and cybernetics-pakt a : systems and humans*, 26 (3) :487–493.

[Leclercq, 2010] Leclercq, J.-P. (Année académique 2009-2010). *Syllabus du cours d'"Optimisation Combinatoire".* Facultés Universitaires Notre-Dame de la Paix, Namur.

[Logothetis and Wandell, 2004] Logothetis, N. K. and Wandell, B. A. (2004). Interpreting the bold signal. *Annual Review of Physiology*, 66 :735–769.

[Maus, 2011] Maus, B. (2011). *Optimal experimental designs for functional magnetic resonance imaging.* PhD thesis, Universitaire Pers Maastricht.

[Michalewicz, 1992] Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs.* Berlin : Springer-Verlag.

[Michel et al., 2002] Michel, S., Michel, B., Adrien, D., Ivor, G.-G., Olivier, H., Rémi, L., Alain, M., Jean, M., and André, R. (2002). *De la méthode : Recherches en histoire et philosophie des mathématiques.* Presses universitaires de Franche-Comté (PuFC) Besançon.

[Miezin et al., 2000] Miezin, F. M., Maccotta, L., Ollinger, J. M., Petersen, S. E., and Buckner, R. L. (2000). Characterizing the hemodynamic response : Effects of presentation rate, sampling procedure, and the possibility of ordering brain activity based on relative timing. *NeuroImage*, 11 :735–759.

[Mitchell, 2009] Mitchell, M. (2009). *Complexity : A Guided Tour.* Oxford University Press.

[Oliver et al., 1987] Oliver, I., Smith, D., and Holland, J. C. (1987). A study of permutation crossover operators on the tsp. In *Proc. Int. Conf. Genetic Algorithms and Their Applications*, pages 224–230. Cambridge, MA.

[Pirlot and Teghem, 2003] Pirlot, M. and Teghem, J. (2003). *Résolution de problèmes de RO par les métaheuristiques.* Lavoisier.

[Poldrack et al., 2010] Poldrack, R. A., Mumford, J. A., , and Nichols, T. E. (2010). *Handbook of functional MRI data analysis.* Cambridge University Press.

[Srinivas and Deb, 1995] Srinivas, N. and Deb, K. (1995). Multiobjective function optimization using nondominated sorting genetic algorithms. *Evol. Comput.*, 2 (3) :221–248.

[Strodiot, 2009] Strodiot, J.-J. (Année académique 2008-2009). *Syllabus du cours d'"Optimisation et Contrôle"*. Facultés Universitaires Notre-Dame de la Paix, Namur.

[Teghem and Pirlot, 2002] Teghem, J. and Pirlot, M. (2002). *Optimisation approchée en recherche opérationnelle*. Lavoisier.

[Wager and Nichols, 2003] Wager, T. D. and Nichols, T. E. (2003). Optimization of experimental design in fmri : a general framework using a genetic algorithm. *NeuroImage*, 18 :293–309.

[Wager et al., 2005] Wager, T. D., Vazquez, A., Hernandez, L., and Noll, D. C. (2005). Accounting for nonlinear bold effects in fmri : Parameter estimates and a model for prediction in rapid event-related studies. *NeuroImage*, 25 :206–218.

[Yeşilyurt et al., 2008] Yeşilyurt, B., Ugurbil, K., and Uludag, K. (2008). Dynamics and nonlinearities of the bold response at very short stimulus durations. *Magn Reson Imaging*, 27 (7) :853–862.